AFRL-OSR-VA-TR-2013-0176

IAPD: Integrated Adaptive and Proactive Defense against Stealthy Botnets

Xu, Shouhuai (PI)
University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249

28-12-2012

Final Report

DISTRIBUTION A: Distribution approved for public release.

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/RTB1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 28-12-2012	**2. REPORT TYPE** Final Report	**3. DATES COVERED** *(From - To)* 3/1/2009-11/30/2012

4. TITLE AND SUBTITLE
IAPD: Integrated Adaptive and Proactive Defense against Stealthy Botnets

5a. CONTRACT NUMBER
210-857-0160

5b. GRANT NUMBER
FA9550-09-1-0165

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Xu, Shouhuai (PI)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research (AFOSR/RSL)
Suite 325, Room 3112
875 N. Randolph Street
Arlington, VA 22203-1768

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION A

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This project studies how to combat stealthy botnets and malwares by exploring a novel framework called IAPD, which stands for “Integrated Adaptive and Proactive Defenses." To achieve the goal, we take a systems-and-theory methodology, meaning that on one hand, we want to build systems that can deal with stealthy attacks, and on the other hand, we want to build a theoretical and foundational understanding of botnets. Such a theoretical understanding allows us to pave the way for achieving principled modeling, management, and decision-making in cyber defense. For systems research, we have built a real-life malware behavior system called Online Malware Analysis System (OMAS), which is under significant further enhancements in design and implementation towards a practical tool. For theoretical research, we have been building mathematical models for understanding and reasoning the attack-defense interactions in cyberspace.

15. SUBJECT TERMS
Bots, Malware, Cyber Defense, Cybersecurity,

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*

Reset

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

Final Technical Report

IAPD: Integrated Adaptive and Proactive Defense against Stealthy Botnets

Grant number: FA9550-09-1-0165

PI: Shouhuai Xu
Department of Computer Science
University of Texas at San Antonio
email: shxu@cs.utsa.edu
webpage: www.cs.utsa.edu/~shxu
phone: 210-458-5739
fax: 210-458-4437

Report date: December 28, 2012

Project period: March 2009 — November 2012
Report period: March 2009 — November 2012

Contents

1 Project Summary

Botnets have become a leading cyber threat because the attackers can use Command & Control (C&C) to launch sophisticated attacks. Despite many studies dealing with botnets, there is a dearth of effective defenses against them, as evidenced by their continued pervasiveness. This project aims to combat stealthy botnets using a novel framework we call IAPD, which stands for "Integrated Adaptive and Proactive Defenses." By "proactive," we mean that the defenses can deal with both current and future botnets and that they do not solely rely on signature-based countermeasures (otherwise, the defenders will always lag behind the attackers). By "adaptive," we mean that the defenses can deal with a spectrum of botnets, from less stealthy to extremely stealthy, in an automated fashion. By "integrated," we mean that the adaptive and proactive defenses are seamlessly integrated together, while being able to utilize information provided by other defensive systems. We believe that the framework represents a necessary step towards a holistic solution.

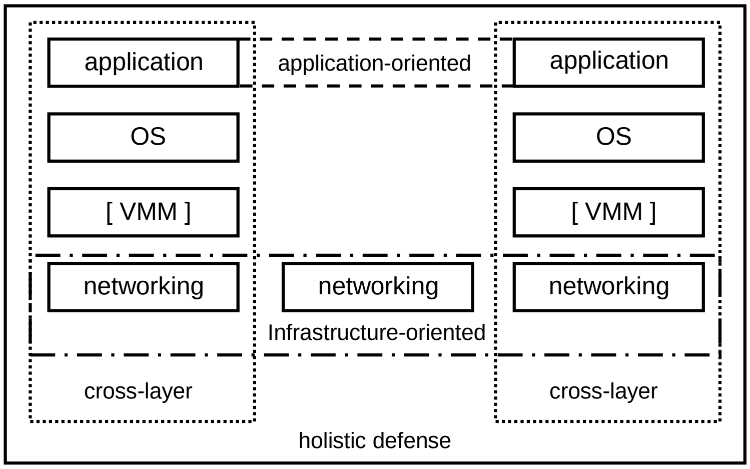


Figure 1: A systematic architecture for defending against bots, botnets and malware (including Advanced Persistent Threats)

To achieve the goal, we take a systems-and-theory methodology, meaning that on one hand, we want to build systems that can deal with stealthy attacks, and on the other hand, we want to build a theoretical and foundational understanding of (defenses against) botnets in particular and malware attacks in general. Such theoretical understanding will pave the way for achieving principled modeling, management, and decision-making in cyber defense. For systems research, we have built real-life stealthy bot/malware analysis and detection systems with the architecture shown in Figure 1, which includes application-oriented, infrastructure-oriented, and cross-layer detections that are seamlessly integrated together. We will elaborate our research activities and results from these perspectives. For theoretical/foundational research, we have built mathematical models for understanding and reasoning the attack-defense dynamics in cyberspace. As we will see, we have obtained some exciting results, which shed light on future research directions that can lead to revolutionary results in cybersecurity.

1.1 Highlight of Findings

Here we highlight some of the findings as follows.

- There are many bots/malwares that can evade the COTS anti-malware detection tools. **Next generation cyber defense technology will have to be simultaneously *behavior-based*, *adaptive* and *proactive*.** We have made substantial progress on these fronts [7, 2, 8, 9, 11, 10, 12, 16, 15]. Our future studies aim to systematically characterize the distinguishing features of malwares based on our large malware database. This will guide us to design the next generation malware/bot defense technology.

- In order to quantify the (in)effectiveness of cyber defense, we need a theoretical foundation. Our studies showed that for this purpose, **multiple disciplines can serve as useful modeling tools**, including Stochastic Processes [17, 22, 18], Dynamical Systems [23, 25, 22, 21], Control Theory [24], Statistical Physics [13, 14, 20] and Statistics [26]. We will further our studies toward a unified foundation. Our envisioned foundation will become on indispensable cornerstone of the emerging Science of Security. In particular, we found that existing mathematical/statistical/physical theories are *not* sufficient for cybersecurity; instead, we need to develop new theories that are guided by, and centered on, cybersecurity problems (i.e., "mathematical theories with cybersecurity meanings/significance").

- We made several conceptual advancements. For example, there are two classes of malware-based attacks: push-based and pull-based [15, 16, 23]. Systematic characterization of these classes of attacks will deepen our understanding of cybersecurity. For this purpose, we introduced some core abstractions for understanding cybersecurity, including stochastic cyber attack processes [26], stochastic cyber attack-defense processes [17, 18], cyber attack-defense dynamics [23, 25, 22, 21]. Finally, cyber attacks and cyber attack-defense interactions exhibit rich phenomena [26, 22, 21]. This highlights the **importance of studying the phenomenon-perspective of cybersecurity (or cybersecurity phenomena)**, which was not recognized until now.

- Our study touched several deepest concepts in cybersecurity. For example, we showed for the first time that cyber attack traffic exhibits the so-called Long-Range Dependence [26], which was previously known to be exhibited by benign traffic. This hints that it is perhaps **impossible to detect (stealthy) attacks based on traffic alone**. This sheds lights on the ***detectability* of cyber attacks**. On the other hand, we showed that Long-Range Dependence can be exploited to predict cyber attacks at a reasonable accuracy [26]. This sheds light on the ***predictability* of cyber attacks**. It would be of fundamental value to characterize the interaction (or trade-off) between detectability and predictability of cyber attacks.

The above suggests exciting research directions that will lead to revolutionary results.

Report outline. Section 2 reports our research results. Section 3 describes our Technology Transfer activities. In Section 4 discusses the DoD relevance of the research results. Appendix A is the copies of some of the papers that acknowledge the research grant. Appendix B is the reports of the subcontracts of the supplementary equipment grant.

2 Research Results

In this Section we describe our research results, which are classified into two categories: systems research and theoretical/foundational research. The two categories of studies are complementary to each other.

2.1 Systems Research

We report our results on characterizing the incapabilities of the current generation of COTS malware/bot defense technologies. Our main results are some novel approaches to the next generation malware/bot defense technologies, including (1) mechanism-level countermeasures against stealthy malwares/bots from an application-oriented, host-oriented, and infrastructure-oriented perspectives, and (2) architecture-level designs for incorporating the novel mechanisms into a seamlessly integrated framework. We also discuss our prototype system for behavior-centric detection of stealthy malware. We are in the process of substantially enhancing this prototype system, which will have a great potential for commercial success.

2.1.1 Limitations of State-of-the-Art COTS Malware/Bot Defenses Technology

Evaluating Detection and Treatment Effectiveness of COTS Anti-Malware Programs. Commercial anti-malware programs consist of two main components: detection and treatment. Detection accuracy is often used to rank effectiveness of commercial anti-malware programs with less emphasis on the equally important treatment component. Effectiveness measures of commercial anti-malware programs should consider equally detection and treatment. This can be achieved by standardized measurements of both components. This paper [11] presents a novel approach to evaluate the effectiveness of a commercial anti-malware program's detection and treatment components against malicious objects by partitioning true positives to incorporate detection and treatment. This new measurement is used to evaluate the effectiveness of four commercial anti-malware programs in three tests. The results show that several anti-malware programs produced numerous incorrectly treated or untreated true positives and false negatives leaving many infected objects unresolved and thereby active threats in the system. These results further demonstrate that our approach evaluates the detection and treatment components of commercial anti-malware programs in a more effective and realistic manner than currently accepted measurements which primarily focus on detection accuracy.

Evaluating the Limitations of Commercial Anti-Malware Tools. Commercial anti-malware programs (CAmps) have become mainstream security products and are widely deployed. In practice, perhaps due to economic factors, most users only purchase and deploy a single anti-malware program. Since it is known and proven that there is no universally effective anti-malware program (which can determine whether a given software program is a malware), this effectively bases malware defense on the implicit assumption that at least the popular anti-malware programs can provide sufficient security. This assumption, surprisingly, has been neither justified nor examined in a systematic fashion. Motivated by our observation obtained in our last-year study that our anomaly-based detection can recognize malwares/bots that were not detected by commercial anti-malware detection tools, in [12] we propose a methodological framework for examining this assumption. We define a CAmp to be competent when it detects and cleans all malware present on a system. Our initial experiments demonstrate that a single CAmp may not suffice. It is also not clear yet how many CAmps are needed to achieve competence in the majority of malware scenarios, though we attempt to forecast a suitable amount based on our experimental results. Our ongoing study aims to resolve this issue to a satisfactory level.

2.1.2 Novel Application-Oriented Malware/Bot Defense Mechanisms

Defense against Social Network-based Stealthy Botnets Command-and-Control. This is an important aspect that has not been paid due attention. Very recently, botmasters have begun to exploit social network websites (e.g., `Twitter.com`) as their C&C infrastructures, which turns out to be quite stealthy because it is hard to distinguish the C&C activities from the normal social networking traffic. This is caused by that attackers can "legitimately" abuse applications that require little or no authorization, which means that the resulting malicious activities are perfectly camouflaged into the traffic of honest users. In [2], we studied the problem of using social networks as botnet C&C infrastructures. Treating as a starting point the current generation of social network-based botnet C&C, we envision the evolution of such C&C methods (which may become real in the near future) and explore social networks-based countermeasures. Specifically:

- We characterize the current-generation of social network-based botnet C&Cs, describing their strengths and weaknesses.

- We envision how current social network-based botnet C&Cs might evolve in the near future, which capitalize on their strengths while diminishing their weaknesses.

- We explore countermeasures for dealing with both current and future generations of social network-based botnet C&Cs. Since social network providers as well as client machines are victims of a social network-based botnet C&C, both server-side and client-side countermeasures are demonstrated and tested for both effectiveness and performance.

2.1.3 Novel Host-Oriented Cross-Layer Malware/Bot Defense Mechanisms

Unlike traditional approaches, we want to achieve (1) behavior-based rather than signature-based detection of malwares/bots, and (2) high resolution rather than low resolution detection of malwares/bots, namely that we want to detect the malicious processes running on a machine rather than just pointing out which machines are compromised.

Differentiating Malicious Programs and Benign Programs Through the Process Lens. In [9], we presented three sets of process-based symptoms drawn from known malware and bot samples — bot network activity behavior, unreliable provenance and stealth mechanisms — that are integrated together to detect bot processes on a host machine. Specifically, we accomplish the following:

- The process-based identification of (1) Bot network activity behavior: failed TCP connection attempts, DNS and reverse DNS queries; (2) Process provenance: using static file image digital signature verification and process/file system tampering; (3) Stealth mechanisms: using the absence of a graphical user interface and no required user input to execute.

- A formal detection model based on a non-trivial use of established data mining algorithms. We conducted a thorough experiment on generating and evaluating detection models. Results show our methodology leads to better detection accuracy for both centralized and Peer-to-Peer (P2P) bots than a straightforward use of established data mining algorithms.

Constructing Malware Infection Trees. Disinfecting unknown malware from a system is a challenging task. A malware infection tree (MiT) can assist a user in disinfecting by terminating processes and deleting files related to the malware. In [10], we propose an approach to constructing a comprehensive MiT based on execution event rules. These rules abstractly define malware infection strategies in files and processes. We implement the rules in a Windows OS tool named MiTCoN which was used to remove malware files and processes resulting in a significant malware eradication from the system. We conclude our rules are strong enough to cover the most important infection strategies and usable in helping disinfect a system compromised by both known and unknown malware.

Cross-Layer Detection of Malicious Websites. Web threats pose the most significant cyber threat. Websites have been developed or manipulated by attackers for use as attack tools. Existing malicious website detection techniques can be classified into the categories of static and dynamic detection approaches, which respectively aim to detect malicious websites by analyzing web contents, and analyzing run-time behaviors using honeypots. However, existing malicious website detection approaches have technical and computational limitations to detect sophisticated attacks and analyze massive collected data. The main objective of this research [15] is to minimize the limitations of malicious website detection. This paper presents a novel cross-layer malicious website detection approach which analyzes network-layer traffic and application-layer website contents simultaneously. Detailed data collection and performance evaluation methods are also presented. Evaluation based on data collected during 37 days shows that the computing time of the cross-layer detection is **50 times faster** than the dynamic approach while detection can be almost as effective as the dynamic approach. Experimental results indicate that the cross-layer detection outperforms existing malicious website detection techniques.

2.1.4 Novel Infrastructure-Oriented Malware/Bot Defense Mechanisms

Differentiating Malicious Programs and Benign Programs Through the DNS Lens. In [7], we investigated a bot process's DNS activities, including DNS and Reverse DNS (rDNS) queries. More specifically, we focused on the process's reaction-to-DNS-response behavior (RD behavior) occurring in the initial join phase during the early stages of a bot process's life cycle. Our detection approach of RD behavior can be implemented at any point in a bot's life cycle; we choose to focus on the early stage because we want to prevent damage and malware distribution in the host machine and network. This is possible because during the initial join phase, bots may frequently use DNS activity to assist in locating their Command & Control (C&C) servers or other peers. We organized different paths of RD behavior that can occur in the join phase as a directed tree, classifying expected versus anomalous, and thereby suspicious, RD behavior. We analyzed five currently active centralized and P2P bots, benign network applications and non-bot malware. During analysis, we identified suspicious RD behavior. We compared our analysis of two commercial bot detectors and combined the results to improve detection accuracy. In summary, the contributions of this research are:

- Identify a novel vector of suspicious process behavior based on the process's reaction-to-DNS-response (RD behavior). We further represent suspicious behavior via paths on a directed tree of DNS activity combined with RD behavior.
- Enhance host-based detection methods with a new vector of detection, namely suspicious RD behavior. We target bot processes rather than just bot machines.
- We show that this suspicious behavior often occurs in the early stages of bot execution, thus detection at this point in time can prevent the bot from executing received commands.

Differentiating Malicious Programs and Benign Programs Through the Network Lens. In [8], we systematically address the following questions: From a networking perspective, do malicious programs (malware, bots, viruses, etc.) behave differently from benign programs that run daily for various needs? If so, how may we exploit the difference in network behavior to detect them? To address these questions, we are systematically analyzing the behavior of a large set (at the magnitude of 2,000) of malware samples. This research analyzes known malicious and benign samples in an attempt to exploit differences in their network behavior to accomplish accurate behavior based malware detection.We present our initial results after analyzing 1000 malware samples. The results show that malicious and benign programs behave quite differently from a network perspective. We are still in the process of understanding the differences, which nevertheless have been utilized to **detect 31 malware samples which evaded detection of all antivirus software on Virustotal.com on 01 April 2010**, giving evidence that the differences between malicious and benign network behavior has a possible use in helping stop zero-day attacks.

2.1.5 Novel Next Generation Malware/Bot Defense Architecture and System

Integrated Adaptive and Proactive Defense. Cyber threats against clouds are dynamic and evolve rapidly. This means that traditional reactive defense is far from sufficient. In the present chapter [6], we advocate that proactive defense should be widely deployed because it offers the defender situation awareness and early warning capabilities in combating against the dynamic cyber threats. Specifically, we describe a security architecture for proactively hunting for new and possibly zero-day attacks. The architecture consists of a set of seamlessly integrated systematic security mechanisms at the application layer, the network layer, and the system layer. Features of the architecture include: (i) it is centered on proactive defense; (ii) it facilitates that a new attack detected at one layer may lead to countermeasures that can be deployed at the other layers; (iii) it uses command and control (C2) centers to coordinate both in-cloud and cross-cloud defense activities. Integrating adaptive and proactive defense requires the support of trustworthy sensing or situation awareness [19].

Proactive, Cross-Layer, Resilient Detection of Malicious Websites. Malicious websites are an important infrastructure of Internet criminal activities. The resulting attacks are often stealthy and more difficult to detect. There has been broad interest in developing systems to prevent end users from visiting such sites. In [16], we investigate the rapid detection of malicious websites by, among other things, tracking URL redirections and conducting content-based analysis. Specifically, we examine websites collected by our client honeypot and other data sources. We demonstrate the usefulness of cross application-network layer security analysis. We characterize conditions under which our detection techniques can or cannot be evaded. An automated software tool for this purpose is under development.

Fighting Through: Tolerating Penetration of Stealthy Malwares. In [1], we proposed and investigated a novel solution to tolerating the penetration of stealthy malware into computer systems, with an emphasis on the protection of cryptographic trustworthiness that *cannot* be resolved by (for example) smart cards. Digital signatures are an important mechanism for ensuring data trustworthiness via source authenticity, integrity, and source nonrepudiation. However, their trustworthiness guarantee can be subverted in the real world by sophisticated attacks, which can obtain cryptographically legitimate digital signatures without actually compromising the private signing key. This problem cannot be adequately addressed by a purely cryptographic approach, by the revocation mechanism of Public Key Infrastructure (PKI) because it may take a long time to detect the compromise, or by using tamper-resistant hardware because the attacker does not need to compromise the hardware. This problem will become increasingly more important and evident because of stealthy malware (or Advanced Persistent Threats). In this paper, we propose a novel solution, dubbed Assured Digital Signing (ADS), to enhancing the data trustworthiness vouched by digital signatures. In order to minimize the modifications to the Trusted Computing Base (TCB), ADS simultaneously takes advantage of trusted computing and virtualization technologies. Specifically, ADS allows a signature verifier to examine not only a signatures cryptographic validity but also its system security validity that the private signing key and the signing function are secure, despite the powerful attack that the signing application program and the general-purpose Operating System (OS) kernel are malicious. The modular design of ADS makes it application-transparent (i.e., no need to modify the application source code in order to deploy it) and almost hypervisor-independent (i.e., it can be implemented with any Type I hypervisor). To demonstrate the feasibility of ADS, we report the implementation and analysis of an Xen-based ADS system.

A Prototype System of Next Generation Behavior-Centric Malware/Bot Defense System. In order to detect *stealthy* and *new* (zero-day) malware, we need anomaly-based detection systems. The proposed project will develop a novel anomaly-based detection system. The system is based on malware-like symptoms and can detect new malware based on their execution behavior, while incurring low (acceptable) false-positives. The symptoms are extracted from our systematic study of a large set of malware samples, at the magnitude of $10^5 - 10^6$. This is made possible by our *automated* malware behavior analysis engine on the Microsoft Windows platform, which is currently called Online Malware Analysis System (OMAS, `http://omas.ics.utsa.edu/`) and released for public testing. Compared with other host-based behavior-based malware analysis systems, OMAS is unique in that it can detect, via our own behavior-based malware detection algorithms, whether the program in question is malicious or not. In other words, OMAS can not only answer the question "what a program in question does to a system" (which is also answered by other comparable analysis systems), but also answer the question "whether the program in question is malicious" (which is not offered by other comparable analysis systems). The current version of OMAS has the following symptoms:

- Self-reference replication (srr) is a fundamental characteristic of computer viruses and worms. This is used as a symptom which our tests indicate is detectable in several malware samples. These samples are broad and diverse. We define srr to be a process which self-replicates into a newly created or pre-existing file. In our testing, srr has proven a powerful early indicator of possible malicious behavior. Dynamic code injection (DCI) is an anomalous behavior extensively used by malware to manipulate running processes into performing nefarious actions. We define DCI as a process writing code commands into the allocated memory of some other already running process and then creating a new instance of that process which executes the just written nefarious code. This symptom has proven highly useful in early malware detection as the vast majority of our tested malware samples have multiple identifiable DCI instances.

- We have created two more symptoms by combining DNS and TCP events which, in our testing have been identified in a good number of known malware samples. We assume a successful DNS query returns an IP address that can be successfully connected to. We also assume the input IP address of a successful Reverse DNS (rDNS) query can also be successfully connected to. Based on this we define as anomalous, and therefore suspicious behavior, a failed TCP connection attempt to the input IP of successful rDNS or to the returned IP of a successful DNS query. Our testing has found multiple instances of these two symptoms. We have established that when these symptoms appear in malware, it is a result of a remote host having been shut down or temporarily removed. We have observed malware perform many DNS events where some are identified under these two symptoms and others follow our notion of normal, non-anomalous, behavior.

- Digital signatures, though not a behavior, are a file attribute that we have blended into our symptoms to assess suspicion. We believe that objects with verifiable digital signatures are less suspicious than those without. In our testing the vast majority of malware samples either completely lack a digital signature or have an unverifiable signature. We find this attribute to be very useful in malware identification.

We are in the process of enhancing the current design and implementation of the prototype, with an emphasis on enhancing the malware/bot behavior-database, which will replace the current COTS solutions that are based on malware/bot signatures.

2.2 Theoretical and Foundational Research

Systems research aims to enhance defense and security. In order to evaluate the situation and the effectiveness of defenses, we need to have a principled foundation. Towards this goal, we have made the following substantial progresses.

2.2.1 Characterizing Botnet Resilience

Modeling and Understanding Botnet. In [4], we presented a model for understanding botnets. The model captures botnet lifetime cycles, botnet architecture, botnet command-and-control (C&C) mechanisms, and a set of botnet attributes. In particular, the model offers insights into how some advanced techniques — such as anonymous communication channels, secret handshakes, and gossiping, which might not have been (widely) implemented in existing botnets — could be adopted in future botnets to make them more difficult to deal with. The model suggests a dynamic graph abstraction to abstract botnets, and the set of botnet attributes can be used to measure and compare botnets. The attributes proposed are: *robustness*, *resilience*, *sustainability*, *exposedness*, *bandwidth consumption*, *botnet size*, *botnet master goals*, and *botnet firepower*. The attributes can be combined to capture properties; for example, the often mentioned term "stealth" can be reflected by what we call robustness, resilience, sustainability, exposedness, and bandwidth consumption.

Graph-Theoretic Model of Botnet C&C. In [3], we proposed a graph-based model for botnet C&C mechanisms. The model captures an important aspect of C&C mechanisms — who knows whom and who can send or forward C&C messages to whom. The model also accommodates that the botnet topology itself might have already "embedded" some craftiness of the botnet master (i.e., botnet topologies are carefully chosen by the masters). Moreover, the model can accommodate the master's attack power or sophistication as well as the defender's detection capability. The model considers two botnet stealth measures called *detection ratio* and *resilience*. The former aims to capture the detection of bots due to the conducting of C&C activities, and the latter aims to capture the tracing of bots based on botnet topology. Simulation study allows us to draw useful insights on factors contributing to botnet stealth, such as the following. First, in order to achieve the same degree of security, countering a more sophisticated attack requires a corresponding improvement in the defender's detection capability. Second, in- and out-degree regular graphs, in which each vertex has the same in-degree as well as out-degree, as botnet topology exhibit best observed stealth. In particular, such graphs exhibit an "all or nothing" detection effect, meaning (1) either all or no bots are detected and (2) the defender cannot benefit from tracing bots according to the botnet topology. Moreover, a botnet master who is able to maintain such a botnet topology does not, in contrast to a recent rule of thumb, necessarily gain stealth by splitting a large botnet into smaller ones.

What Can Statistical Physics Contribute to Cybersecurity? Site percolation has been used to help understand analytically the robustness of complex networks in the presence of random node deletion (or failure). In [13], we move a further step beyond random node deletion by considering that a node can be deleted because it is chosen or because it is within some L-hop distance of a chosen node. Using the generating functions approach, we present analytic results on the percolation threshold as well as the mean size, and size distribution, of nongiant components of complex networks under such operations. The introduction of parameter L is both conceptually interesting because it accommodates a sort of nonindependent node deletion, which is often difficult to tackle analytically, and practically interesting because it offers useful insights for cybersecurity (such as botnet defense). Deeper results are obtained in a MS Thesis [14], which is split into papers for publications. In particular, we found that *emergent behavior* is ubiquitous in cybersecurity [20].

2.2.2 Understanding Effectiveness of Next Generation Malware/Bot Defense Technology

Modeling and Characterizing the Usefulness of Adaptive Control in Cyber Defense. In [24], we investigate a non-homogeneous Susceptible-Infectious-Susceptible (SIS) model in arbitrary networks (i.e., there is no restriction on the topology of the spreading networks and the nodes may have different defense or cure capabilities). The model can accommodate both *semi-adaptive* defense and *fully-adaptive* defense. In the semi-adaptive defense scenario, the input parameters in the model are known and can vary with respect to time (e.g., according to some deterministic functions of time or according to some stochastic process, but we do not impose any practical restrictions on the types of functions). For this scenario, we present a set of sufficient conditions, from general to specific (but more succinct), under which the virus spreading will die out. We note that such sufficient conditions are also known as *epidemic thresholds* in the literature. In the fully-adaptive defense scenario, some input parameters are not known and thus the aforementioned sufficient conditions are not applicable. Nevertheless, the defender might be able to observe the outcome of virus spreading (i.e., which nodes are infected at a point in time). For this scenario, we present **adaptive control** strategies under which the virus spreading will die out or will be contained to a desired level (which is important when, for example, the price to kill the virus spreading may be too high). Because of the above, our model supersedes previous homogeneous and non-homogeneous models that offered relevant analytical insights. Our analytical results are confirmed via simulation, from which we draw additional observations that serve as hints for future modeling studies. We discuss the practical implications of our model and the derived insights as well.

Characterizing the Advantage of Active Defense over Reactive Defense. Current cyber defense is mainly reactive, and inevitably causes an asymmetry that is to the advantage of the attacker. One approach to circumventing the asymmetry is to exploit active cyber defense. Although the concept of active cyber defense has been proposed and debated for years, the focus has been primarily on policy-related issues rather than on characterizing its effectiveness. In [22], we initiate the theoretical study on characterizing the effectiveness of active cyber defense. For this purpose, we propose a novel Markov model to accommodate the interaction between cyber attacks and active defense. Since the Markov model is not tractable because of its exponential number of states, we simplify it as a Dynamic System model, which leads to useful analytical results and insights. We use simulation to validate the analytical results by showing that they are inherent to the Markov model. In [21], we extend our study to more general active defense dynamics, which actually exhibits richer phenomena such as Bifurcation and Chaos. We discuss their fundamental implications for cybersecurity (e.g., measurability and predictability).

Modeling and Characterizing Push- and Pull-based Epidemic Spreading in Networks: Thresholds and Deeper Insights. In [23], we investigate a general cyber attack-defense dynamics model that accommodate, in arbitrary networks, both *push-based* infection, namely that a compromised node always actively attempts to attack its neighboring nodes and *pull-based* infection, which accommodates attacks such as "drive-by download" attack (e.g., a vulnerable computer getting compromised after connecting to a compromised website). This paper significantly deepens our understanding of push- and pull-based epidemic spreading dynamics in arbitrary networks. Specifically, our main contributions are: (1) We present a general sufficient condition (also known as epidemic threshold) under which the spreading becomes stable. The threshold supersedes the existing ones derived in the push-based infection models. (2) We give both upper and lower bounds on the global mean infection rate; the bounds are not always tight but do allow us to draw interesting and useful observations. (3) When the spreading is stable, we draw the following deeper insights into: the role of node degree in governing node infection rate; conditions under which the popular mean field approach can be used in arbitrary networks; estimating the global mean infection rate through localized monitoring of a *small* number of nodes *without* knowing the values of the parameters and *independent of* the size of the network.

Modeling and Characterizing Multi-Virus Dynamics. Understanding the spreading dynamics of computer viruses (worms, attacks) is an important research problem, and has received much attention from the communities of both computer security and statistical physics. However, previous studies have mainly focused on *single-virus* spreading dynamics. In [25], we study *multi-virus* spreading dynamics, where multiple viruses attempt to infect computers while possibly combating against each other because, for example, they are controlled by multiple botmasters. Specifically, we propose and analyze a general model (and its two special cases) of multi-virus spreading dynamics in arbitrary networks (i.e., we do not make any restriction on network topologies), where the viruses may or may not co-reside on computers. Our model offers analytical results for addressing questions such as: What are the sufficient conditions (also known as epidemic thresholds) under which the multiple viruses will die out? What if some viruses can "rob" others? What characteristics does the multi-virus epidemic dynamics exhibit when the viruses are (approximately) equally powerful? The analytical results make a fundamental connection between two types of factors: defense capability and network connectivity. This allows us to draw various insights that can be used to guide security defense.

2.2.3 Quantitative Cybersecurity: Towards the Science of Cyber Security

Quantitative Cybersecurity Models without Making the Poisson Assumption. Quantitative security analysis of networked computer systems has been an open problem in computer security for decades. Recently, a promising approach was proposed in [5], which, however, made some strong assumptions including the exponential distribution of, and the independence among, the relevant random variables. In [17], we substantially weaken these assumptions while offering, in addition to the same types of analytical results as in [5], methods for obtaining the desired security quantities in practice. Moreover, we investigate the problem from a higher-level abstraction, which also leads to both analytical results and practical methods for obtaining the desired security quantities. These should represent a significant step toward ultimately solving the problem of quantitative security analysis of networked computer systems.

Quantitative Cybersecurity Models without Making the Independence Assumption. Models of epidemics over arbitrary complex networks have a great potential to become one of the pillars in the emerging foundation of cyber security. However, this will not happen until after we have resolved a set of challenges. A particular challenge is to tame the dependence in cyber epidemic process/dynamics models. Due to its notorious difficulty, essentially all existing relevant cyber epidemic models assumed away the dependence between the relevant random variables. In [18], we move a significant first step towards ultimately taming the dependence challenge. Specifically, we present an epidemic (more precisely, attack-defense) process model for cyber security. The model is based on a generalization of a state-of-the-art cyber attack-defense dynamics model that was introduced in [5] and thoroughly investigated in [23]. Our approach to tackling dependence is based on Copulas, but our main results are obtained without assuming any specific Copula structures (i.e., they hold for arbitrary and unknown dependence structures). We also characterize the price of assuming away the due dependence in cyber epidemic models.

Rich Phenomena Exhibited by Cyber Attacks: Data-driven Statistical Characterization of Cyber Attacks. In [26], we report the Long-Range Dependence (LRD) phenomenon that is exhibited by honeypot-captured cyber attacks. To our knowledge, LRD was not known to be relevant in the cyber security domain until now, although it has been known for two decades that LRD is exhibited by benign traffic (i.e., no attacks). We demonstrate the cyber security implications of the LRD phenomenon, by showing how to predict the rates of incoming attacks with reasonable good precision, especially at the scale of "aggregation" the victims as a network. This implies the possibility of providing real-life defenders with sufficient early-warning time for adjusting their defense configurations (e.g., proactively allocating more resources for deep packet inspection).We also explore the cause of the LRD phenomenon. We find that the theory, which was developed for explaining the possible cause of LRD in benign traffic, is probably not applicable to the LRD observed in the cyber security domain. We further find that LRD observed in the cyber security domain is unlikely caused by the intensity of attacks that are launched by individual attackers.

3 Technology Transfer to, and Collaboration with, AFRL

Through AFRL researcher Dr. Keesook Han, the project delivered:

- Tens of thousands malware samples to AFRL.
- Daily updated list of malicious websites to AFRL as well as MITRE.
- Two joint provisional patent applications [16] in collaboration with AFRL researchers Dr. Keesook Han and Mr. Frank Born. These works were partly done when the PI (Dr. Shouhuai Xu) was visiting AFRL/Rome in the summer of 2011 (under AFOSR SFFP).

The supplementary equipment grant included five subcontractors. All the subcontractors have benefited from the equipment grant, which will continue benefiting their research and collaboration with AFRL researchers. The reports of the subcontracts are attached in Appendix B.

4 DoD Interest

The research results are of high DoD interests. On one hand, the systems research results paved the way for designing and realizing the next generation malware/bot (or more generally, cyber) defense technologies. Such technologies can be directly deployed to protect DoD cyber systems, with particular emphasis on combating stealthy malwares/bots, zero-day attacks, and Advanced Persistent Threats. On the other hand, the theoretical/foundational research made substantial progresses towards the Science of Security, which is the Holy Grail challenge and has extremely high relevance to DoD missions. For example, such foundational understanding and models can lead to real-time decision-making tools for cyber operations and mission assurance.

References

[1] Weiqi Dai, Paul Parker, Hai Jin and Shouhuai Xu. *Enhancing Data Trustworthiness via Assured Digital Signing.* IEEE Transactions Dependable Secure Computing 9(6): 838-851 (2012).

[2] Erhan J. Kartaltepe, Jose Andre Morales, Shouhuai Xu, and Ravi Sandhu. *Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures.* Proceedings of the 2010 Applied Cryptography and Network Security (ACNS'10), pp 511-528.

[3] Justin Leonard, Shouhuai Xu, and Ravi S. Sandhu. *A First Step towards Characterizing Stealthy Botnets.* In Proceedings of the The Forth International Conference on Availability, Reliability and Security (ARES'09), pp 106-113.

[4] Justin Leonard, Shouhuai Xu, and Ravi S. Sandhu. *A Framework for Understanding Botnets.* In Proceedings of the 2009 International Workshop on Advances in Information Security, in conjuction with the The Forth International Conference on Availability, Reliability and Security (ARES'09), pp 917-922.

[5] Xiaohu Li, Paul Parker, and Shouhuai Xu. *A Stochastic Model for Quantitative Security Analysis of Networked Systems.* IEEE Transactions on Dependable and Secure Computing, 8(1): 28-43 (2011).

[6] Weiliang Luo and Li Xu and Zhenxin Zhan and Qingji Zheng and Shouhuai Xu. *A Security Architecture for Situation Awareness and Early Warning of Dynamic Cyber Threats against Clouds.* Book chapter to appear in 2013 Springer book entitled "High Performance Semantic Cloud Audit".

[7] Jose Andre Morales, Areej Al-Bataineh, Shouhuai Xu, and Ravi Sandhu. *Analyzing DNS Activities of Bot Processes.* In Proceedings of the 4th International Conference on Malicious and Unwanted Software (Malware'09).

[8] Jose Andre Morales, Areej Al-Bataineh, Shouhuai Xu, and Ravi Sandhu. *Analyzing and Exploiting Network Behaviors of Malware.* Proceedings of SecureComm'2010, pp 20-34.

[9] Jose Andre Morales, Erhan Kartaltepe, Shouhuai Xu and Ravi Sandhu. *Symptoms-Based Detection of Bot Processes.* Proceedings of the 5th International Conference on Mathematical Methods, Models, and Architectures for Computer Networks Security (MMM-ACNS-2010), pp 229-241.

[10] Jose Andre Morales, Michael Main, Weiliang Luo, Shouhuai Xu, and Ravi Sandhu. *Building Malware Infection Trees.* Proceedings of the 6th International Conference on Malicious and Unwanted Software (Malware 2011).

[11] Jose Morales, Ravi Sandhu, and Shouhuai Xu. *Evaluating Detection and Treatment Effectiveness of Commercial Anti-Malware Programs.* Proceedings of the 5th International Conference on Malicious and Unwanted Software (Malware 2010).

[12] Jose Morales, Shouhuai Xu, and Ravi Sandhu. *Analyzing Malware Detection Efficiency with Multiple Anti-Malware Programs*. Proceedings of The 1st ASE International Conference on Cyber Security, 2012.

[13] Yilun Shang, Weiliang Luo, and Shouhuai Xu. *L-hop percolation on networks with arbitrary degree distributions and its applications*. Physical Review E, Sept. 2011.

[14] Adam Tyra. A Characterization of Complex Network Attack Resilience. MS Thesis supervised by the PI, University of Texas at San Antonio, 2012.

[15] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. *Cross-Layer Detection of Malicious Websites*. Proceedings of the Third ACM Conference on Data and Application Security and Privacy (ACM CODASPY'13).

[16] Li Xu, Zhenxin Zhan, Shouhuai Xu, Keying Ye, Frank Born and Keesook Han. *Resilient Cross-Layer Detection of Malicious Websites*. Two patent provisional applications were filed in June 2012. Technical Report, Approved for Public Release; Distribution Unlimited: 88ABW-2011-5874, 04 Nov 2011.

[17] Maochao Xu and Shouhuai Xu. *An Extended Stochastic Model for Quantitative Security Analysis of Networked Systems*. Internet Mathematics, 8(3): 288-320 (2012).

[18] Maochao Xu and Shouhuai Xu. *Towards Taming the Dependence in Cyber Attack-Defense Processes*, to be submitted, 2012.

[19] Shouhuai Xu. *Towards a Theoretical Framework for Trustworthy Cyber Sensing*. In Proceedings of the 2010 SPIE Conference on SPIE Defense, Security, and Sensing (DSS'10).

[20] Shouhuai Xu. *Emergent Behavior in Cybersecurity*. to be submitted, 2012.

[21] Shouhuai Xu and Wenlian Lu. *Rich Phenomena of Active Cyber Defense Dynamics*. to be submitted, 2012.

[22] Shouhuai Xu, Weiliang Lu and Hualun Li. *A Stochastic Model of Active Cyber Defense*. Under journal review, 2012.

[23] Shouhuai Xu, Wenlian Lu, and Li Xu. *Push- and Pull-based Epidemic Spreading in Networks: Thresholds and Deeper Insights*. ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS), 7(3): 32 (2012).

[24] Shouhuai Xu, Wenlian Lu, Li Xu, and Zhenxin Zhan. *Adaptive Epidemic Dynamics in Networks: Thresholds and Control*. ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS) special issue on Adaptive Distributed Defense Systems, accepted for publication.

[25] Shouhuai Xu, Wenlian Lu, and Zhenxin Zhan. *A Stochastic Model of Multi-Virus Dynamics*. IEEE Transactions on Dependable and Secure Computing (IEEE TDSC), 9(1): 30-45 (2012).

[26] Zhenxin Zhan, Maochao Xu and Shouhuai Xu. *Long-Range Dependence as Exhibited by Honeypot-Captured Cyber Attacks: The Phenomenon, Implication and Mysterious Cause*. In submission, 2012.

5 Appendix A: Copy of 10 Selected Papers

Copies of the following 10 papers are attached:

1. Li Xu, Zhenxin Zhan, Shouhuai Xu, Keying Ye, Frank Born and Keesook Han. *Resilient Cross-Layer Detection of Malicious Websites.* Two patent provisional applications were filed in June 2012. Technical Report, Approved for Public Release; Distribution Unlimited: 88ABW-2011-5874, 04 Nov 2011.

2. Maochao Xu and Shouhuai Xu. *An Extended Stochastic Model for Quantitative Security Analysis of Networked Systems.* Internet Mathematics, 8(3): 288-320 (2012).

3. Yilun Shang, Weiliang Luo, and Shouhuai Xu. *L-hop percolation on networks with arbitrary degree distributions and its applications.* Physical Review E, Sept. 2011.

4. Shouhuai Xu, Wenlian Lu, and Li Xu. *Push- and Pull-based Epidemic Spreading in Networks: Thresholds and Deeper Insights.* ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS), 7(3): 32 (2012).

5. Shouhuai Xu, Wenlian Lu, Li Xu, and Zhenxin Zhan. *Adaptive Epidemic Dynamics in Networks: Thresholds and Control.* ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS) special issue on Adaptive Distributed Defense Systems, accepted for publication.

6. Shouhuai Xu, Wenlian Lu, and Zhenxin Zhan. *A Stochastic Model of Multi-Virus Dynamics.* IEEE Transactions on Dependable and Secure Computing (IEEE TDSC), 9(1): 30-45 (2012).

7. Zhenxin Zhan, Maochao Xu and Shouhuai Xu. *Long-Range Dependence as Exhibited by Honeypot-Captured Cyber Attacks: The Phenomenon, Implication and Mysterious Cause.* In submission, 2012.

8. Erhan J. Kartaltepe, Jose Andre Morales, Shouhuai Xu, and Ravi Sandhu. *Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures.* Proceedings of the 2010 Applied Cryptography and Network Security (ACNS'10), pp 511-528.

9. Jose Andre Morales, Areej Al-Bataineh, Shouhuai Xu, and Ravi Sandhu. *Analyzing and Exploiting Network Behaviors of Malware.* Proceedings of SecureComm'2010, pp20-34.

10. Jose Morales, Michael Main, Weiliang Luo, Shouhuai Xu, and Ravi Sandhu. *Building Malware Infection Trees.* Proceedings of the 6th International Conference on Malicious and Unwanted Software (Malware 2011).

RESILIENT CROSS-LAYER DETECTION OF MALICIOUS WEBSITES: TOWARDS ACHIEVING THE BEST OF BOTH STATIC AND DYNAMIC ANALYSES AND GOING BEYOND

Li Xu[1], Zhenxin Zhan[1], Shouhuai Xu[1], Keying Ye[2], Frank Born[3], Keesook Han[3]
[1] Department of Computer Science, University of Texas at San Antonio
[2] Department of Statistics, University of Texas at San Antonio
[3] Air Force Research Laboratory, Information Directorate

Abstract

Malicious websites have become a major attack tool of the adversary. Detection of malicious websites in real-time can facilitate early-warning and filtering the contents from, and the accesses to, the malicious websites. There are two main approaches to detecting malicious websites: *static* and *dynamic*. The static approach is centered on the analysis of website contents, and thus can automatically detect malicious websites in a very efficient fashion and can scale up to a large number of websites. However, this approach has limited success in dealing with sophisticated attacks that include obfuscation. The dynamic approach is centered on the analysis of website contents via their run-time behavior, and thus can cope with these sophisticated attacks. However, this approach is often expensive and cannot scale up to the magnitude of the number of websites in cyberspace. In this paper, we propose a novel cross-layer solution that can inherit the advantages of the static approach while overcoming its drawbacks. Our solution is centered on the following: (i) application-layer web contents, which were typically analyzed in the static approach, may not provide sufficient information for detection; (ii) network-layer traffic corresponding to application-layer communications might provide extra information that can be exploited to substantially enhance the detection of malicious websites. Evaluation of our cross-layer detection is based on real-life data that we collected. In order to deal with attacks that attempt to evade our cross-layer detection, we investigate the resilience of our solution against adaptive attacks. We demonstrate that adaptive attacks can easily evade detections that include our own, and propose algorithms for effectively defending against adaptive attacks.

Approved for Public Release; Distribution Unlimited: 88ABW-2011-5874, 04 Nov 2011

RESILIENT CROSS-LAYER DETECTION OF MALICIOUS WEBSITES: TOWARDS ACHIEVING THE BEST OF BOTH STATIC AND DYNAMIC ANALYSES AND GOING BEYOND

Li Xu[1], Zhenxin Zhan[1], Shouhuai Xu[1], Keying Ye[2], Frank Born[3], Keesook Han[3]
[1] Department of Computer Science, University of Texas at San Antonio
[2] Department of Statistics, University of Texas at San Antonio
[3] Air Force Research Laboratory, Information Directorate

1 Introduction

Malicious websites have become a severe cyber threat because they can cause the automatic download and execution of malware in browsers, and thus compromise vulnerable computers [43]. The phenomenon of malicious websites will persevere at least in the near future because we cannot prevent websites from being compromised or abused. Existing approaches to detecting malicious websites can be classified into two categories:

- The *static* approach aims to detect malicious websites by analyzing their URLs [31, 32] or their contents [51]. This approach is very efficient and thus can scale up to deal with the huge population of websites in cyberspace. This approach however has trouble coping with sophisticated attacks that include obfuscation [47], and thus can cause high false-negative rate by classifying malicious websites as benign ones.
- The *dynamic* approach aims to detect malicious websites by analyzing their run-time behavior using Client Honeypots or their like [50, 53, 4, 3]. Assuming the underlying detection is competent, this approach is very effective. This approach however is resource-consuming because it runs or emulates the browser and possibly the operating system [11]. As a consequence, this approach cannot scale up to deal with the large number of websites in cyberspace.

Because of the above, it has been advocated to use a front-end light-weight tool, which is mainly based on static analysis and aims to rapidly detect suspicious websites, and a back-end more powerful but much slower tool, which conducts a deeper analysis of the detected suspicious websites.

While conceptually attractive, the success of this *hybrid* approach fundamentally relies on the assumption that the front-end static analysis does have very low false-negative rates; otherwise, many malicious websites will not be detected even if the back-end dynamic analysis tools are powerful. However, this assumption can be easily violated because of the following: First, in real life, the attacker could defeat pure static analysis by exploiting various sophisticated techniques such as obfuscation and redirection. Second, the attacker can get the same data and therefore use the same machine learning algorithms to derive the defender's classifiers. This is plausible because in view of Kerckhoffs's Principle in cryptography, we should assume that the defender's learning algorithms are known to the attacker. As a consequence, the attacker can always act one step ahead of the defender by adjusting its activities so as to evade detection. The above two issues lead to the following question: how can we achieve the best of both static and dynamic analysis, and go beyond? The question is important and motivates the investigation presented in this paper.

Our contributions:

Towards the ultimate goal of fully addressing the above two issues, in this paper we propose a novel solution to the problem of detecting malicious websites. At a high-level, our solution analyzes the website contents as well as the redirection website contents in the fashion of the static approach, while taking advantage of the network-layer traffic information. More specifically, our contributions are the following. First, we use *static* analysis to proactively track redirections, which have been abused by the attacker to hide malicious websites. Our study shows that static analysis can be extended to track redirections and detect many malicious websites.

Second, we exploit the network-layer traffic information to gain significant extra detection capabilities. For example, we find that network-layer J48 classifier [45] achieves 99.91% detection accuracy, 0.47% false-negative rate and 0.03% false-positive rate. They are significantly better than their application-layer counterpart, which achieves 98.99% detection accuracy, 7.63% false-negative rate and 0.03% false-positive rate. Moreover, using the PCA (Principle Component Analysis [24]) feature selection method that leads to 80 features that are used in the classifier training process, cross-layer J48 classifier achieves 99.94% detection accuracy, 0.06% false-negative rate, and 0.05% false-positive rate. With the feature selection method called `GainRatioAttributeEval` [24], cross-layer J48 classifier achieves 99.91% detection accuracy, 0.477% false-negative rate, and 0.03% false-positive rate. This is a somewhat surprising result because the training process is based on 9 out of the 124 features in total. Among the 9 features, 5 are application-layer ones and 4 are network-layer ones. This suggests that the data collection system can be made much more efficient because it only needs to collect a few critical features.

Third, we present three adaptation algorithms that may be used by the attacker to launch adaptive attacks, and can be exploited by the defender to launch adaptive defense. Our algorithms allow us to demonstrate how easy it can be for an adaptive attacker to evade non-adaptive detection. We also show how our algorithms can effectively deal with adaptive attacks, and thus make our detection system resilient to adaptive attacks. This is true even if a few features are used for training J48 classifiers.

Evaluation of our solution is based on real data, except for resilience analysis where we manipulate real data to mimic adaptive attacks. We have developed a detection system that serves as the fast and scalable front-end of a comprehensive solution, which is depicted in Figure 1.

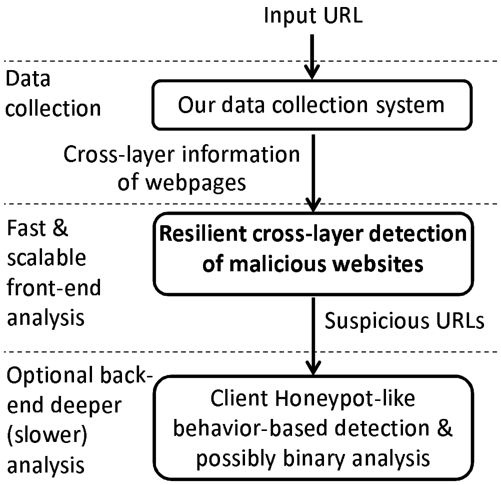

Input URL
Data collection
Our data collection system
Cross-layer information of webpages
Fast & scalable front-end analysis
Resilient cross-layer detection of malicious websites
Suspicious URLs
Optional back-end deeper (slower) analysis
Client Honeypot-like behavior-based detection & possibly binary analysis

Figure 1: The relationship between our detection system and a comprehensive solution.

Paper organization:

The rest of the paper is organized as follows. Section 2 describes our cross-layer data collection and analysis. Section 3 investigates the resilience of our cross-layer detection system. Section 4 discusses related prior work. Section 5 elaborates the limitations of our system and suggests future research directions. Section 6 concludes the paper.

Summary of main notations:

The following table summarizes the main notations used in the paper.

X_z	the zth feature of feature vector X
$\min_z, \max_z$	$\min$ and $\max$ value of the zth feature
$M_i(D_j)$	applying classifier M_i to dataset D_j in adaptive attack setting
$M_{0\text{-}i}(D_j)$	the majority vote of $M_0(D_j)$, $M_1(D_j)$, ..., $M_i(D_j)$ in adaptive attack setting
f, g, h	functions the attacker uses to manipulate its behavior in adaptive attack setting

2 Cross-layer Analysis and Detection of Malicious Websites

In this section, we describe the architecture of our cross-layer data collection system, and present our analysis methodology and the evaluation results of our classifiers.

2.1 Cross-layer Data Collection and Pre-processing

Data collection method and system architecture:

In order to facilitate cross-layer analysis and detection, we built an automated system to collect both the application-layer URL contents and the resulting network-layer traffic. The architecture of our automated data collection system is depicted in Figure 2. At a high-level, our data collection system is centered on a crawler, which takes a list of URLs as input, automatically fetches the website contents by launching HTTP/HTTPS requests to the target URLs, and tracks the redirections it identified from the website contents (elaborated below). The crawler further uses the URLs, including both the input ones and the resulting redirection URLs, to query the DNS, Whois, and Geographic services for collecting relevant features for analysis. The application-layer web contents and the corresponding network-layer IP packets are recorded separately, but are indexed by the input URLs to facilitate cross-layer analysis. The collected application-layer raw data are pre-processed to make them suitable for machine learning tasks (also elaborated below).

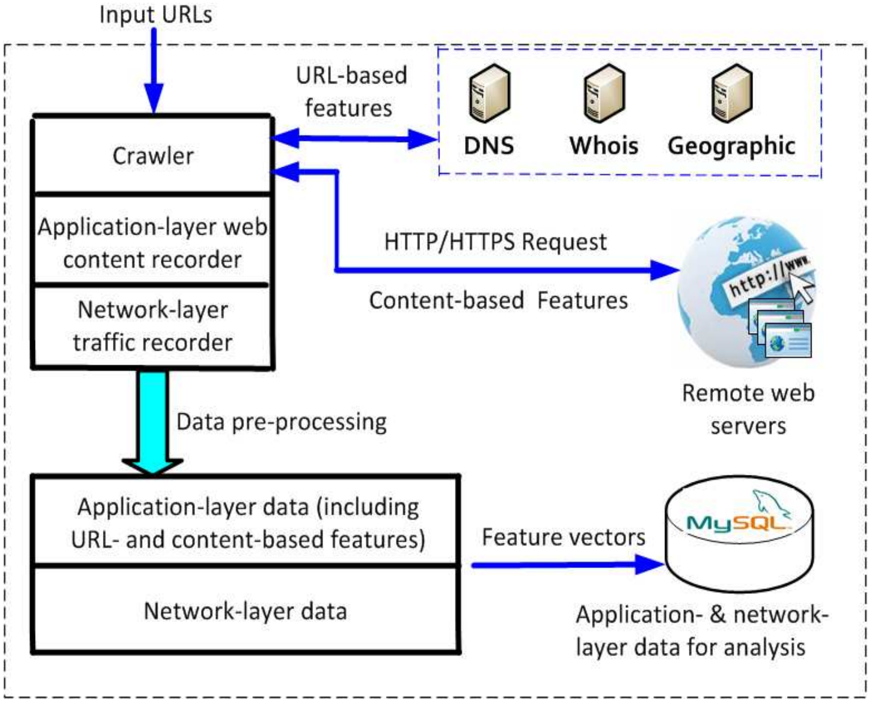

Input URLs
URL-based features
Crawler
DNS
Whois
Geographic
Application-layer web content recorder
HTTP/HTTPS Request
Network-layer traffic recorder
Content-based Features
http://www
Remote web servers
Data pre-processing
Application-layer data (including URL- and content-based features)
Feature vectors
MySQL
Network-layer data
Application- & network-layer data for analysis

Figure 2: Data collection system architecture.

Statically and proactively tracking redirects:

The data collection system proactively tracks redirections by analyzing the website contents in a static fashion, which means that our solution is as fast and scalable as the static approach. Specifically, we consider the following four types of redirects. The first type is server side redirects that are initiated either by server rules (i.e., `.htaccess` file) or server side page code such as `PHP`. These redirects often utilize HTTP 300 level status codes. The second type is JavaScript-based redirects. Despite extensive study, there has been limited success in dealing with JavaScript-based redirection that is coupled with obfuscation [19]. The third type is the refresh Meta tag and HTTP refresh header, which allow one to specify the URLs of the redirection pages. The fourth type is embedded file redirects. Examples of this type are: `<iframe src='badsite.php'/>`, `<img src='badsite.php'/>`, and `<script src='badsite.php' > </script>`.

It is important to understand that the vast majority of malicious URLs are actually victim sites that have themselves been hacked. Sophos Corporation has identified the percentage of malicious code that is hosted on hacked sites as 90% [5]. Most often this malicious code is implanted using SQL injection methods and shows up in the form of an embedded file as identified above. In addition, stolen ftp credentials allow hackers to have direct access to files where they can implant malicious code directly into the body of a web page. The value of the embedded file method to hackers is that, through redirections and changing out back end code and file references, they can better hide the malicious nature of these embedded links from search engines and browsers.

Description and pre-processing of application-layer data:

The resulting application-layer data have 105 features in total, which are obtained after pre-processing the collected application-layer raw data. The application-layer raw data consist of feature vectors that correspond to the respective input URLs. Each feature vector consists of various features, including information such as HTTP header fields, and information obtained by using the input URLs and the detected redirection URLs to query DNS and Whois services. In particular, redirection information includes (i) redirection method, (ii) whether a redirection points to a different domain, (iii) the number of redirection hops.

Because different URLs may involve different numbers of redirection hops, different URLs may have different numbers of features. This means that the application-layer raw feature vectors do not necessarily have the same number of features, and thus cannot be processed by both classifier learning algorithms and classifiers themselves. We resolve this issue by aggregating multiple-hop information into *artificial* single-hop information as follows: for numerical data, we aggregate them by using their average instead; for boolean data, we aggregate them by taking the `OR` operation; for nominal data, we only consider the final destination URL of the redirection chain. For example, suppose that an input URL is redirected twice to reach the final destination URL and the features are (`Content-Length`, "Does JavaScript function `eval()` exist in the code?", `Country`). Suppose that the raw feature vectors corresponding to the input, first redirection, and second redirection URLs are (100, `FALSE`, US), (200, `FALSE`, UK), and (300, `TRUE`, RUSSIA), respectively. We aggregate the three raw feature vectors as (200, `TRUE`, RUSSIA), which is stored in the application-layer data for analysis.

Description of network-layer data:

Network-layer features are extracted from the corresponding PCAP (Packet CAPture) files that are recorded when the crawler accesses the input URLs. There are 19 network-layer features that are derived from IP level, UDP/TCP level or flow level, where a flow is uniquely identified by a tuple (source IP, source port #, destination IP, destination port #, protocol). Some examples of network-layer features are: `Iat_flow`, which is the cumulative inter-arrival time between the flows caused by the access to an input URL; `DNS_query_times`, which is the total number of DNS queries caused by the access to an input URL; `TCP_conversation_exchange`, which is the number of conversation exchanges in the TCP connections caused by the crawler's access to an input URL; `IP_packets`, which is the number of IP packets caused by the access to an input URL; `Avg_remote_rate`, which is the rate the remote server sends to local host (packets per second) during the crawler's access to an input URL.

2.2 Cross-layer Data Analysis Methodology

Classifier accuracy metrics:

Suppose that the defender learned a classifier M from some training data. Suppose that the defender is given test dataset D, which consists of d_1 malicious URLs and d_2 benign URLs. Suppose further that among the d_1 malicious URLs, M correctly detected d_1' of them, and that among the d_2 benign URLs, M correctly detected d_2' of them. The *detection accuracy* of M is defined as $\frac{d_1'+d_2'}{d_1+d_2}$. The *false-positive rate* is defined as $\frac{d_2-d_2'}{d_2}$. The *true-positive rate* is defined as $\frac{d_1'}{d_1}$. The *false-negative rate* is defined as $\frac{d_1-d_1'}{d_1}$. Ideally, we want a classifier to achieve high detection accuracy, low false-positive rate and low false-negative rate.

Data analysis methodology:

Our analysis methodology was geared towards answering questions about the power of cross-layer detection. It has two steps, which are equally applicable to both application-layer and network-layer data.

1. Feature selection (optional): Because there are 124 features in total, we may need to conduct feature selection. We used the following three feature selection methods. The first method is "`PCA` with the Ranker search method" (`PCA` for short) that transforms a set of feature vectors to a set of shorter feature vectors [24].

   The second feature selection method is called "`CfsSubsetEval` with the Best-First search method" in the Weka toolbox (`CfsSubsetEval` for short) [24]. It essentially computes the features' prediction power, and its selection algorithm essentially ranks the features' contributions [23]. It outputs a subset of features that are substantially correlated with the class (`benign` or `malicious`) but have low inter-feature correlations.

   The third feature selection method is called "`GainRatioAttributeEval` with the Ranker search method" (`GainRatioAttributeEval` for short) in the Weka toolbox [24]. Its evaluation algorithm essentially computes the information gain ratio (or more intuitively the importance of each feature) with respect to the class, and its selection algorithm ranks features based on their information gains [16]. It outputs the ranks of all features in the order of decreasing importance.

2. Model learning and validation: We used four popular learning algorithms: Naive Bayes, Logistic, SVM, and J48, which have been implemented in the Weka toolbox [24]. Naive Bayes classifier is a probabilistic classifier based on Bayes' rule [25]. Logistic classifier [29] is one kind of linear classification, where the domain of the target variable is $\{0, 1\}$. SVM (Support Vector Machine) classifier aims to find an maximum-margin hyperplane for separating different classes in the training data [14]. We use the SMO (Sequential Minimal-Optimization) algorithm in our experiment with polynomial kernel function, which is efficient [42]. J48 classifier is the Weka implementation of C4.5 decision trees [45], which can be used for binary classification.

For cross-layer data analysis, we consider the following two cross-layer aggregation methods.

- Data-level aggregation. The application-layer feature vector and the network-layer feature vector with respect to the same URL are simply merged into a single longer feature vector. This is possible because the vectors are indexed by the input URLs. In this case, the data-level aggregation operation is conducted before the above two-step data analysis process.
- Model-level aggregation. The decision on whether a website is malicious is based on the decisions of the application-layer classifier and the network-layer classifier. There are two options. One option is that a website is classified as malicious if the application-layer classifier *or* the network-layer classifier says it is malicious; otherwise, it is classified as benign. We call this OR-aggregation. The other option is that a website is classified as malicious if both the application-layer classifier *and* the network-layer classifier say it is malicious; otherwise, it is classified as benign. We call this AND-aggregation. In either case, both application- and network-layer data are processed using the above two-step data analysis process. Then, the resulting classifiers are further aggregated using the OR or AND operation.

Datasets description:

Our dataset consists of 1,467 malicious URLs and 10,000 benign URLs. The malicious URLs are part of the 28,018 blacklist URLs downloaded from malware.com.br, malwaredomainlist.com and compuweb.com/url-domain-bl.txt on the same day as we conducted the experiment (Nov. 3, 2011), and are confirmed as malicious by the high-interaction client honeypot Capture-HPC version 3.0 [50]. Our test of blacklist URLs using high-interaction client honeypot confirmed our observation that some or many currently blacklisted URLs are not accessible any more and thus should not be counted as malicious URLs. The 10,000 benign URLs are obtained from alexa.com, which lists the top 10,000 websites that are supposed to be well protected.

2.3 On the Power of Cross-Layer Detection

In order to identify the more powerful classifiers, we compare the aforementioned four classifiers, with and without feature selections. Table 1 describes the results without using feature selection, using the PCA and CfsSubsetEval feature selections. We make the following observations. First, by taking a horizontal perspective, we observe that for cross-layer detection with or without using feature selection, J48 classifier often performs better than the other three classifiers. Moreover, for data-level aggregation, J48 classifier achieves the highest detection accuracy and the lowest false-negative rate. For OR-aggregation, J48 classifier achieves the highest detection accuracy and the lowest false-positive rate. For AND-aggregation, J48 classifier naturally leads to the lowest false-positive rate, but also causes a relatively high false-negative rate.

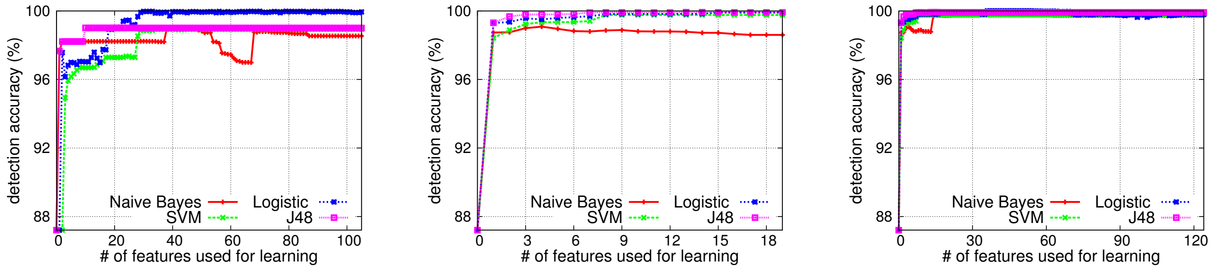
Second, by taking a vertical perspective, we observe that J48 classifier based on data-level aggregation and without using feature selection achieves the highest detection accuracy and the lowest false-negative rate. Given that we have singled out data-level aggregation and OR-aggregation cross-layer J48 classifier, let us now look at whether using feature selection will jeopardize classifier quality. We observe that using PCA feature selection actually leads to roughly the same, if not better, detection accuracy, false-negative rate, and false-positive rate.

In the case of data-level aggregation, J48 classifier can be trained using 80 features that are derived from the 124 features using PCA; the CfsSubsetEval feature selection method actually leads to the use of four network-layer features: (1) Local_app_bytes, which is the accumulated application bytes of TCP packets sent from local host to the remote server. (2) Dist_remote_tcp_port, which is the cumulative number of distinct TCP ports that have been used by the remote server. (3) Iat_flow, which is the accumulated inter-arrival time between flows. (4) Avg_remote_rate, which is the rate the remote server sends to local host (packets per second). This can be explained as follows: malicious websites that contain malicious JavaScript code or contents can cause frequent and large volume communications between remote servers and local hosts. Note that this does not contradict the fact that our crawler is a static analysis tool. This is because our crawler program only downloads the source code of web

Layer	Feature selection method	Naive Bayes			Logistic			SVM			J48		
		Acc.	FN	FP	Acc.	FN	FP	Acc.	FN	FP	Acc.	FN	FP
Application-layer	none	98.54	11.31	0.01	99.87	0.27	0.1	98.92	7.43	0.13	98.99	7.63	0.03
	PCA	94.44	1.43	6.16	99.76	1.64	0.04	99.60	2.93	0.02	99.88	0.68	0.03
	CfsSubsetEval	98.45	4.23	1.16	99.81	1.30	0.03	99.69	2.38	0.0	99.80	1.29	0.03
Network-layer	none	98.60	1.91	1.32	99.90	0.61	0.03	99.75	1.90	0.0	99.91	0.47	0.03
	PCA	78.09	55.94	9.39	79.94	58.09	6.07	78.44	69.69	3.85	94.88	9.32	3.57
	CfsSubsetEval	77.86	72.25	3.71	80.88	56.89	5.23	77.56	79.48	1.46	95.71	6.77	3.38
Cross-layer (data-level agg.)	none	99.75	1.84	0.01	99.79	0.74	0.12	99.78	1.70	0.0	99.91	0.47	0.03
	PCA	87.32	24.47	10.94	99.61	1.29	0.25	99.41	4.49	0.01	99.94	0.06	0.05
	CfsSubsetEval	98.44	4.22	1.16	99.80	1.29	0.03	99.69	2.38	0.0	99.80	1.29	0.03
Cross-layer (OR-aggregation)	none	98.65	1.50	1.33	99.89	0.00	0.13	99.63	1.91	0.14	99.89	0.48	0.06
	PCA	85.82	1.43	16.05	99.28	1.64	0.59	98.97	2.93	0.75	99.92	0.00	0.09
	CfsSubsetEval	97.97	1.23	2.15	98.63	1.30	1.38	97.61	2.39	2.39	98.97	1.30	0.99
Cross-layer (AND-aggregation)	none	98.50	11.72	0.00	99.89	0.89	0.00	99.05	7.43	0.00	99.02	7.63	0.00
	PCA	91.83	58.55	0.78	98.93	8.38	0.00	98.91	8.52	0.00	99.81	1.50	0.00
	CfsSubsetEval	98.67	10.43	0.00	99.05	7.43	0.00	95.13	38.10	0.00	99.05	7.43	0.00

Table 1: Comparison (%) between no feature selection and two feature selection methods (Acc.: detection accuracy; FN: false-negative rate; FP: false-positive rate).

pages, parses its content but does not execute it. Our crawler is different from traditional web browsers that have a display engine and dynamically execute JavaScript code.



(a) Application-layer

(b) Network-layer

(c) Cross-layer (data-level agg.)

Figure 3: Detection accuracy (%) vs. the number of features selected from training classifiers (in order of decreasing significance). When no features are used, the classifier can class all URLs as benign, which implies 87.21% detection accuracy because there are 87.21% benign URLs.

2.4 On the Feasibility of Using a Few Features for Learning Classifiers

How few features can we use to train classifiers? To answer this question, we use the GainRatioAttributeEval feature selection method because it actually ranks the contributions of the individual features. Figure 3 plots the experimental results, which allow us to make the following observations. For application-layer, using the following eleven features already leads to 99.01% detection accuracy for J48 (98.88%, 99.82%, 99.76% for Naive Bayes, Logistic and SVM, respectively). (1) HTTPHeader_server, which is the type of the HTTP server at the redirection destination of an input URL (e.g., Apache, Microsoft IIS, etc.). (2) Whois_regDate, which is the registration date of the website that corresponds to the redirection destination of an input URL. (3) HTTPHeader_cacheControl, which indicates the cache management method in the server side. (4) Whois_stateProv, which is the registration state or geographical location of the website. (5) Charset, which is encoding charset of current URL (e.g., iso-8859-1), and hints the language a website used and its target users' ethnicity. (6) Within_domain, which indicates whether the destination URL and the original URL are in the same domain. (7) Updated_date, which is the last update date of the final redirection destination website. (8) Content_length_valid, which indicates whether the content length field in the HTTP header is valid. This is relevant because the content length field could be a negative number, which may cause buffer overflow attacks. (9) Redirect_num, which is the total number of redirects embedded into an input URL to destination URL. Malicious websites often have a larger number of redirects than benign websites. (10) Country, which is the country of the registrant. (11) Protocol, which indicates the transfer protocol a website uses. Note that HTTPS is normally used by benign websites. When these 11 features are used for training classifiers, we can achieve detection accuracy of 98.22%, 97.03%, 96.69% and 99.01% for Naive Bayes, Logistic, SVM and J48 classifiers respectively.

For network-layer, using the following nine features can have good detection accuracy and lower false-negative rate. (1) `Avg_remote_packet_rate`, which is the average IP packets rate (packets per second) sent by the remote server. For multiple remote IP addresses, this feature is the average IP packets rate of the remote website. (2) `Dist_remote_TCP_port`, which is the number of distinct TCP ports opened by remote servers. (3) `Dist_remote_IP`, which is the number of distinct remote server IP addresses. (4) `DNS_answer_times`, which is the number of DNS answers sent by DNS server. (5) `Flow_number`, which is the number of flows. (6) `Avg_local_packet_rate`, which is the average rate of IP packets that are sent by the local host (packets per second). (7) `DNS_query_times`, which is the number of DNS queries sent by local host. (8) `Duration`, which is the duration of time consumed for a conversation between the local host and the remote server. (9) `Local_app_packets`, which is the number of IP packets sent by the local host to the remote server. When these nine features are used for training classifiers, we can achieve detection accuracy of 98.88%, 99.82%, 99.76% and 99.91% for Naive Bayes, Logistic, SVM and J48 classifiers respectively. An explanation of this phenomenon is the following: Because of redirection, visiting malicious URLs will cause local host to send multiple DNS queries and connect to multiple remote servers, which might lead to high volume communications.

We observe, as expected, that J48 classifier performs at least as good as the others in terms of network-layer detection and cross-layer detection. Note that in this case we have to compare the false-negative and false-positive rates with respect to the number of features that are used for learning classifiers. In Table 2, we summarize the false-negative and false-positive rates of the classifiers learned from a few features. The five application-layer features and four network-layer features used in the data-level aggregation case are the top five (out of the eleven) `GainRatioAttributeEval`-selected features used by the application-layer classifier and the top four (out of the nine selected) `GainRatioAttributeEval`-selected features used by the network-layer classifier, respectively. The eleven application-layer features and nine network-layer features used in the `OR`-aggregation and `AND`-aggregation are the same as the features that are used in the application-layer and network-layer classifiers. We make the following observations. First, J48 classifier learned from fewer application-layer features, network-layer features and cross-layer features can still maintain very close detection accuracy and false-negative rate.

Layer	number of	Naive Bayes			Logistic			SVM			J48		
	features	Acc.	FN	FP	Acc.	FN	FP	Acc.	FN	FP	Acc.	FN	FP
Application	11	98.22	7.430	0.95	97.04	7.430	2.3	96.69	7.430	2.7	98.21	7.430	0.96
Network	9	98.79	1.908	1.099	99.81	1.226	0.03	99.75	1.908	0.0	99.90	0.545	0.03
Cross (data-level agg.)	5+4	98.88	1.908	1.0	99.81	1.295	0.02	99.75	1.908	0.0	99.91	0.477	0.03
Cross (`OR`-aggregation)	11+9	98.06	1.23	2.05	97.82	1.23	2.32	97.40	1.91	2.70	99.07	0.55	0.99
Cross-layer (`AND`-aggregation)	11+9	98.96	8.11	0.000	99.04	7.43	0.010	99.05	7.43	0.000	99.05	7.43	0.00

Table 2: Effect when a few features are used for learning classifiers (Acc.: detection accuracy; FN: false-negative rate; FP: false-positive rate; $a + b$: a application-layer features plus b network-layer features).

Second, for all the cross-layer classifiers based on data-level aggregation, five application-layer features (i.e., `HTTPHeader_server`, `Whois_regDate`, `HTTPHeader_cacheControl`, `Within_domain`, `Updated_date`) and four network-layer features (i.e., `Avg_remote_packet_rate`, `Dist_remote_TCP_port`, `Dist_remote_IP`, `DNS_answer_times`) can already achieve almost as good as, if not better than, the other scenarios. In particular, J48 actually achieves 99.91% detection accuracy, 0.477% false-negative rate, and 0.03% false-positive rate, which is comparable to the J48 classifier learned from all the 124 features, which leads to 99.91% detection accuracy, 0.47% false-negative rate, and 0.03% false-positive rate without using any feature selection method (see Table 1). This means that data-level aggregation with as few as nine features is practical and highly accurate.

Third, Naive Bayes classifier exhibits the following interesting phenomenon: the detection accuracy actually drops when more features are used for building classifiers. One possible cause of this "off-trend" phenomenon is that the feature selection method was not the Bayesian approach and hence not optimum. We leave it to future work to theoretically explain the cause of this phenomenon.

2.5 On the Practicality of the Cross-layer System

As discussed in the Introduction, we aim to make our system as fast and scalable as the static analysis approach while achieving high detection accuracy, low false-negative rate, and low false-positive rate as the dynamic approach. In the above, we have demonstrated that our cross-layer system, which can be based on either the data-level aggregation or the OR-aggregation and even using as few as nine features in the case of data-level aggregation, achieved high detection accuracy, low false-negative rate, and low false-positive rate. In what follows we confirm that, even without using any type of optimization and collecting all the 124 features rather than the necessary nine features, our system is at least about 25 times faster than the dynamic approach. To be fair, we should note that we did not consider the time spent for learning classifiers and the time spent for applying a classifier to the data collected from a given URL. This is because the learning process only takes respectively 6.4, 1.15, and 9.31 seconds for learning the J48 classifier from the application-layer, network-layer, and data-level aggregation data, and because the process of applying a classifier to a given data takes no more than 1 second.

In order to measure the performance of our data collection system, it would be natural to use the time spent on collecting the cross-layer data information and the time spent by the client honeypot system. Unfortunately, this is not feasible because our data collection system is composed of several computers with different hardware configurations. To resolve this issue, we conducted extra experiments using two computers with the same configuration. One computer will run our data collection system and the other computer will run client honeypot system. The hardware of the two computers is Intel Xeon X3320 4 cores CPU and 8GB memory. We use Capture-HPC client honeypot version 3.0.0 and VMWare Server version 1.0.6, which runs on top of a Host OS (Windows Server 2008) and hosts a Guest OS (Windows XP sp3). Since Capture-HPC is high-interactive and thus necessarily heavy-weight, we ran five guest OS (according to our experiment, more guest OS will make the system unstable), and used default configuration of Capture-HPC.

Our data collection system uses a crawler, which was written in JAVA 1.6 and runs on top of Debian 6.0. Besides the JAVA based crawler, we also use IPTABLES [2] and modified version of TCPDUMP [6] to obtain high parallel capability. When running multiple crawler instances at the same time, the application features can be obtained by each crawler instance, but the network feature of each URL should also be extracted. TCPDUMP software can be used to capture all the outgoing and incoming network traffic on local host. IPTABLES can be configured to log network flow information with respect to processes with different user identity. We use different user identity to run each crawler instance, extract network flow information for each URL and use the flow attributes to extract all the network packets of a URL. Because our crawler is light-weight, we conservatively ran 50 instances in our experiments.

Input URLs	Our crawler	Capture-HPC
Malicious (1,562)	4 min	98 min
Benign (1,500)	4 min	101 min

Table 3: Time comparison between Capture-HPC and our crawler

The input URLs in our performance experiments consist of 1,562 malicious URLs that are accessible, and 1,500 benign URLs that are the first 1,500 websites listed by Alexa.com. Table 3 shows the performance of the two systems. We observe that our crawler is about 25 times faster than Capture-HPC, which demonstrates the performance gain of our system. We note that in the experiments, our cross-layer data collection system actually collected all 124 features. The performance can be further improved if only the necessary features (nine in the above data-level aggregation method) are collected.

Summary:

We demonstrated that cross-layer information leads to better classifiers. We further demonstrated that using as few as nine cross-layer features, including five application-layer features and four network-layer features, the resulting J48 classifier is almost as good as the one that is learned using all the 124 features. We showed that our data collection system roughly can be at least 25 times faster than the dynamic approach based on Capture-HPC.

3 Resilience Analysis against Adaptive Attacks

Cyber attackers often adjust their attacks to evade the defense. In the previous section, we demonstrated that the J48 classifier is very effective. However, it may be possible that the J48 classifier can be easily evaded by an adaptive attacker. In this section, we partially resolve the issue.

Because the problem is fairly complicated, we start with the example demonstrated in Figure 4. Suppose that the attacker knows the defender's J48 classifier M. The leaves are decision nodes with class 0 indicating benign

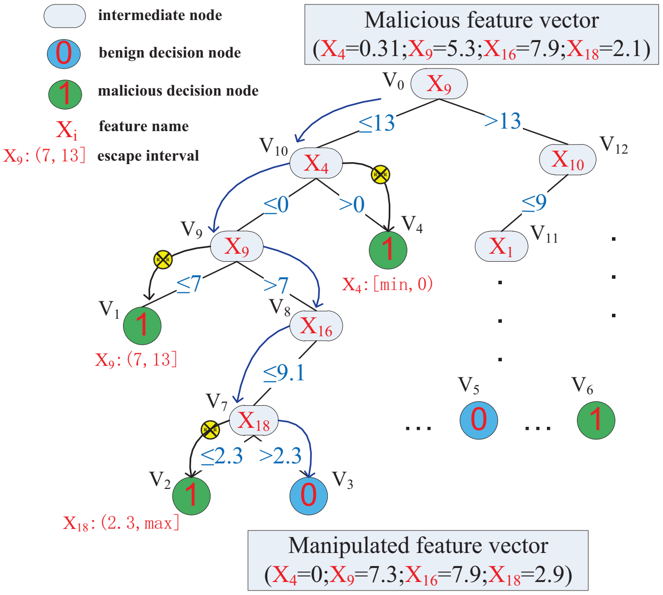


Figure 4: Example manipulation of features to evade J48 classifier.

URL, which is called *benign decision node*, and 1 indicating `malicious` URL, which is called *malicious decision node*. Given the classifier, it is straightforward to see that a URL associated with feature vector $(X_4 = 0.31; X_9 = 5.3; X_{16} = 7.9; X_{18} = 2.1)$, is `malicious` because of the decision path $v_0 \xrightarrow{X_9 \leq 13} v_{10} \xrightarrow{X_4 > 0} v_4$. To evade detection, an adaptive attacker can adjust the URL properties that lead to feature vector $(X_4 = 0; X_9 = 7.3; X_{16} = 7.9; X_{18} = 2.9)$. As a consequence, the URL will be classified as benign because of the decision path $v_0 \xrightarrow{X_9 \leq 13} v_{10} \xrightarrow{X_4 \leq 0} v_9 \xrightarrow{X_9 \leq 7} v_8 \xrightarrow{X_{16} \leq 9.1} v_7 \xrightarrow{X_{18} > 2.3} v_3$. Now the questions are: How can the attacker adjust to manipulate the feature vectors? How should the defender respond to adaptive attacks? As a first step towards a systematic study, in what follows we will focus on a class of adaptive attacks and countermeasures, which are characterized by three adaptation strategies that are elaborated below.

3.1 Resilience Analysis Methodology

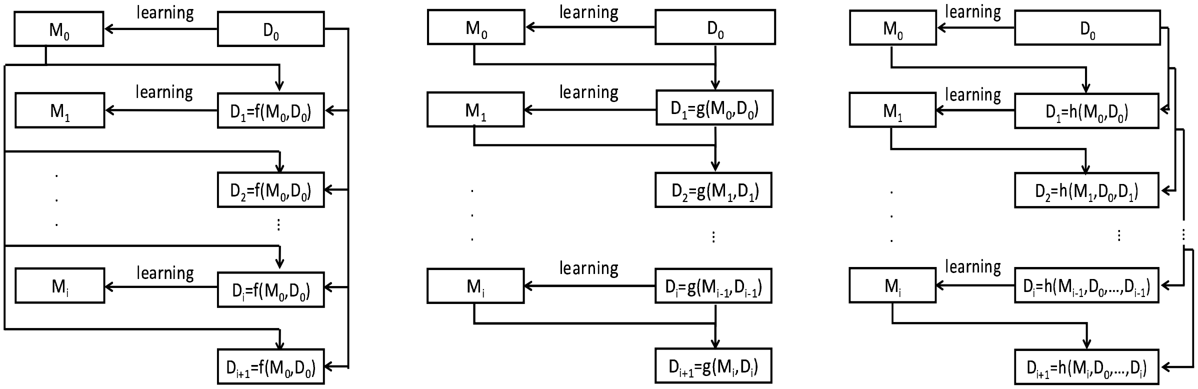
Resilience metrics:

In order to characterize the resilience of cross-layer detection against adaptive attacks, we compare the effect of non-adaptive defense and adaptive defense against adaptive attacks. The effect will be mainly illustrated through the true-positive rate, which intuitively reflects the degree that adaptive attacks cannot evade the defense. The effect will also be secondarily illustrated through the false-positive rate, which reflects the overall quality of the defense. This is because a classifier can achieve high true-positive rate while suffering high false-positive rate.

Three adaptation strategies:

Suppose that system time is divided into epochs $0, 1, 2, \ldots$. The time resolution of epochs (e.g., hourly, weekly, or monthly) is an orthogonal issue and its full-fledged investigation is left for future work. At the ith epoch, the defender may use the collected data to learn classifiers, which are then used to detect attacks at the jth epoch, where $j > i$ (because the classifier learned from the data collected at the current epoch can only be used to detect future attacks at any appropriate time resolution). Since the attacker knows the data collected by the defender and also knows the learning algorithms used by the defender, the attacker can build the same classifiers as the ones the defender may have learned. Given that the attacker always acts one epoch ahead of the defender, the attacker always has an edge in evading the defender's detection. How can we characterize this phenomenon, and how can we defend against adaptive attacks?

In order to answer the above question, it is sufficient to consider epoch i. Let D_0 be the the cross-layer data the defender has collected. Let M_0 be the classifier the defender learned from the training portion of D_0. Because the



(a) Parallel adaptation (b) Sequential adaptation (c) Full adaptation

Figure 5: Three adaptation strategies (f, g, h represent three possibly randomized functions that are chosen by the attacker from respective function families).

attacker knows essentially the same M_0, the attacker may correspondingly adapt its activities in the next epoch, during which the defender will collect data D_1. When the defender applies M_0 to D_1 in real-time, the defender may not be able to detect some attacks whose behaviors are intentionally modified by the attacker to bypass classifier M_0. Given that the defender knows that the attacker may manipulate its behavior in the $(i+1)$st epoch, how would the defender respond? Clearly, the evasion and counter-evasion can escalate further and further. While it seems like a perfect application of Game Theory to formulate a theoretical framework, we leave its full-fledged formal study for future work because there are some technical subtleties. For example, it is infeasible or even impossible to enumerate all the possible *manipulations* the attacker may exploit to evade M_0. As a starting point, we here consider the following three strategies that we believe to be representative.

- **Parallel adaptation:** This strategy is highlighted in Figure 5(a). Specifically, given D_0 (the data the defender collected) and M_0 (the classifier the defender learned from D_0), the attacker adjusts its behavior accordingly so that $D_1 = f(D_0, M_0)$, where f is some appropriately-defined randomized function that is chosen by the attacker from some function family. Knowing what machine learning algorithm the defender may use, the attacker can learn M_1 from D_1 using the same learning algorithm. Because the attacker may think that the defender may know about f, the attacker can repeatedly use f multiple times to produce $D_i = f(M_0, D_0)$ and then learn M_i from D_i, where $i = 2, 3, \ldots$. Note that because f is randomized, it is unlikely that $D_i = D_j$ for $i \neq j$.
- **Sequential adaptation:** This strategy is highlighted in Figure 5(b). Specifically, given D_0 (the data the defender collected) and M_0 (the classifier the defender learned from D_0), the attacker adjusts its behavior so that $D_1 = g(D_0, M_0)$, where g is some appropriately-defined randomized function that is chosen by the attacker from some function family, which may be different from the family of functions from which f is chosen. Knowing what machine learning algorithm the defender may use, the attacker can learn M_1 from D_1 using the same learning algorithm. Because the attacker may think that the defender may know about g, the attacker can repeatedly use g multiple times to produce $D_i = g(M_{i-1}, D_{i-1})$ and then learn M_i from D_i, where $i = 1, 2, \ldots$.
- **Full adaptation:** This strategy is highlighted in Figure 5(c). Specifically, given D_0 and M_0, the attacker adjusts its behavior so that $D_1 = h(D_0, M_0)$ for some appropriately-defined randomized function that is chosen by the attacker from some function family, which may be different from the families of functions from which f and g are chosen. Knowing what machine learning algorithm the defender may use, the attacker can learn M_1 from D_1 using the same learning algorithm. Because the attacker may think that the defender may know about h, the attacker can repeatedly use h multiple times to produce $D_i = h(M_{i-1}, D_0, \ldots, D_{i-1})$ and then learn M_i from D_i, where $i = 1, 2, \ldots$.

Defender's strategies to cope with adaptive attacks:

How should the defender react to the adaptive attacks? In order to characterize the resilience of the classifiers against adaptive attacks, we need to have real data, which is impossible without participating in a real attack-defense escalation situation. This forces us to use some method to obtain synthetic data. Specifically, we design functions f, g, and h to manipulate the data records corresponding to the malicious URLs, while keeping intact the data records corresponding to the benign URLs. Because f, g, and h are naturally specific to the defender's learning algorithms, we here propose the following specific functions/algorithms corresponding to J48, which was shown in the previous section to be most effective for the defender.

At a high-level, Algorithm 1 takes as input dataset D_0 and adaptation strategy ST $\in \{f, g, h\}$. In our case study, the number of adaptation iterations is arbitrarily chosen as 8. This means that there are 9 classifiers $M_0, M_1, \ldots, M_8$, where M_i is learned from D_i.

- For parallel adaptation, we consider the following f function: D_i consists of feature vectors in D_0 that correspond to benign URLs, and the manipulated versions of the feature vectors in D_0 that correspond to the malicious URLs.
- For sequential adaptation, we consider the following g function: D_{i+1} consists of the benign portion of D_0, and the manipulated portion of D_i where the manipulation is conducted with respect to classifier M_i.
- For full adaptation, we consider the following h function: the benign portion of D_{i+1} is the same as the benign portion of D_0, and the manipulated portion is derived from $D_0, D_1, \ldots, D_i$ and D'_i, where D'_i is obtained by manipulating D_i with respect to classifier M_i.

Algorithm 1 Defender's algorithm main(D_0, ST, α)

INPUT: D_0 is the input feature vectors, ST indicates attack strategy, α is the number of adaptation iterations
OUTPUT: $M_0, \ \ldots, \ M_\alpha$ are classifiers

```
initialize array $D_0, \ \ldots, \ D_{\alpha+1}$ where $D_i$ is the feature vectors consisting of benign URLs (dubbed $benFV$) and malicious URLs (dubbed $malFV$)
for $i$=0 to $\alpha$ do
    $M_i \leftarrow$ J48.buildmodel($D_i$)
    switch
    case 1 ST = PARALLEL-ADAPTATION
     $D_i \leftarrow D_i.benFV +$ manipulate($D_0.malFV, M_0$) {this is one example of function $f$}
    case 2 ST = SEQUENTIAL-ADAPTATION
     $D_{i+1} \leftarrow D_i.benFV +$ manipulate($D_i.malFV, M_i$) {this is one example of function $g$}
    case 3 ST = FULL-ADAPTATION
     $D_{i+1}.benFV \leftarrow D_0.benFV$
     $D'_i \leftarrow$ manipulate($D_i.malFV, M_i$)
     $D_{i+1}.malFV \leftarrow \emptyset$
     for $j = 1$ to $malFV.size$
      $d \ \in_R \ D_0.malFV[j], \ \ldots, \ D_i.malFV[j], \ D'_i[j]$
      $D_{i+1}.malFV[j] \leftarrow d$
     end for
    {this is one example of function $h$}
    end switch
end for
return $M_i(i = 0, \ldots, 8)$
```

In order to help understand Algorithm 2, let us consider another example in Figure 4. For each non-leaf node v associated to feature X_i, $v.ival$ is the interval [$\min_i$, condition_value]. For example, in Figure 4, v_0 is associated with feature X_9 and v_0's condition_value is 13, which means $v_0.ival = [\min_9, 13]$. Feature vector $(X_4 = -1; X_9 = 5; X_{16} = 5; X_{18} = 0)$ will lead to decision path $v_0 \xrightarrow{X_9 \leq 13} v_{10} \xrightarrow{X_4 \leq 0} v_9 \xrightarrow{X_9 \leq 7} v_1$, which means that the corresponding URL is classified as malicious. For feature X_9, let us denote its domain by $Domain(X_9) = \{\min_9, \ldots, \max_9\}$, where $\min_9$ ($\max_9$) is the minimum (maximum) value of X_9. In order to evade detection, the attacker can manipulate the value of feature X_9 so that v_1 will not be on the decision path. This can be achieved by assigning a random value from interval $(7, 13]$, which is called *escape interval* and can be derived as

$$([\min_9, \max_9] \setminus [\min_9, 7]) \bigcap [\min_9, 13] \quad = \quad (Domain(X_9) \setminus v_9.ival) \bigcap v_0.ival.$$

Algorithm 2 is based on the above observation and aims to assign *escape interval* to each malicious decision node, which is then used in Algorithm 3.

The basic idea underlying Algorithm 3 is to transform a feature vector, which corresponds to a malicious URL, to a feature vector that will be classified as benign. We use the same example to illustrate how the algorithm works.

Algorithm 2 Algorithm preparation(DT)

```
1:  initiate an empty queue $Q$
2:  for all $v \in DT$ do
3:      if $v$ is leaf AND $v$ = “malicious”  then
4:          append $v$ to queue $Q$
5:      end if
6:  end for
7:  for all $v \in Q$ do
8:      $v.featureName \leftarrow v.parent.featureName$
9:      $v.ival \leftarrow Domain(v.parent) \setminus v.esc\_ival$        $\{Domain(X)$ is the domain of feature $X\}$
10:     $v' \leftarrow v.parent$
11:     while  $v' \neq root$ do
12:         if  $v'.featureName = v.feautreName$ then
13:             $v.esc\_ival \leftarrow v'.ival \bigcap v.esc\_ival$
14:         end if
15:         $v' \leftarrow v'.parent$
16:     end while
17: end for
```

Considering feature vector $(X_4 = -1; X_9 = 5; X_{16} = 5; X_{18} = 5)$, the adaptive attacker can randomly choose a value, say 8, from $v_1.esc\_ival$ and assign it to X_9. This will make the new decision path avoid v_1 but go through its sibling v_8 because the new decision path becomes

$$v_0 \xrightarrow{X_9 \leq 13} v_{10} \xrightarrow{X_4 \leq 0} v_9 \xrightarrow{X_9 > 7} v_8 \xrightarrow{X_{16} \leq 9.1} v_7 \xrightarrow{X_{18} > 2.3} v_3.$$

The key idea is the following: if the new decision path still reaches a malicious leaf, the algorithm will recursively manipulate the feature values of its parent by diverting to its sibling node.

Now let us discuss the time complexity of the defender's algorithms. Recall that α is the number of adaptation times. Suppose that the training dataset has N feature vectors and each feature vector has m features. According to [38], the size of decision tree is $O(N)$. In Algorithm 2, lines 2 to 6 have time complexity $O(N)$, which is to traverse the decision tree and find malicious decision nodes (i.e., leaves with `malicious` label). Lines 7 to 17 have time complexity $O(N^2)$, which is to assign *escape interval* to each malicious decision node. In Algorithm 1, line 3 has time complexity $O(Nm^2)$ [45] and lines 13 to 16 have time complexity $O(N)$. The overall time complexity is $O(\alpha N(m^2 + N))$. In practice, this time complexity is lightweight even on a regular desktop (at the magnitude of tens of seconds).

Evaluation scenarios:

Because there are three aggregation methods and both the attacker and the defender can take the three adaptation strategies, there are $3 \times 3 \times 3 = 27$ scenarios. In the present paper, for each aggregation method, we focus on three scenarios that can be characterized by the assumption that *the attacker and the defender will use the same adaptation strategy*; we will investigate all possible scenarios in the future. For each scenario, we consider the following four configurations:

- The attacker does not adapt but the defender adapts multiple times.
- The attacker adapts once but the defender does not adapt.
- The attacker adapts once but the defender adapts multiple times.
- Both the attacker and the defender adapt multiple times.

3.2 Cross-layer Resilience Analysis

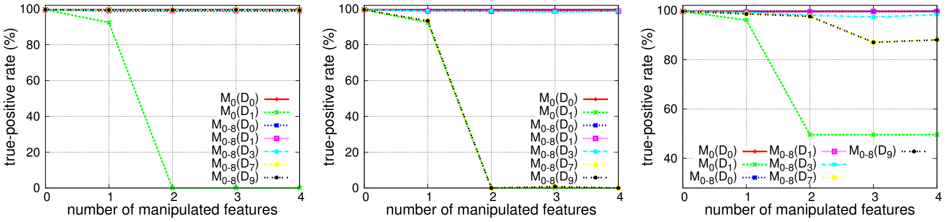
Resilience measurement through true-positive rate:

Figure 6 plots the true-positive rate with respect to the number of features that are manipulated by an adaptive attacker in the case of data-level aggregation. We observe that if the attacker is adaptive but the defender is non-adaptive, then most malicious URLs will not be detected as we elaborate below. For parallel and sequential adaptations, the true-positive rate of $M_0(D_1)$ drops to 0% when the attacker adapts its behavior by manipulating two features. Even in the

Algorithm 3 Algorithm manipulate(D, M) for transforming malicious feature vector to benign feature vector

```
INPUT: $D$ is dataset, $M$ is classifier
OUTPUT: manipulated dataset
$DT \leftarrow M.DT$ { $DT$ is the J48 decision tree}
preparation($DT$)
for all feature vector $fv \in D$ do
  $v \leftarrow DT.root$
  while NOT ($v$ is leaf AND $v \neq$ "$benign$") do
    if $v$ is leaf AND $v =$ "$malicious$" then
      pick a value $n \in v.ival$ at random
      $fv$.setFeatureValue($v.featureName, n$)
      $v \leftarrow v.sibling$
    end if
    if $v$ is not leaf then
      if $fv.featureValue \leq v.featureValue$ then
        $v \leftarrow v.leftChild$
      else
        $v \leftarrow v.rightChild$
      end if
    end if
  end while
end for
return D
```

case of full adaptation defense, the true-positive rate of $M_0(D_1)$ can drop to about 50% when the attacker adapts its behavior by manipulating two features. We also observe that if the attacker is not adaptive but the defender is adaptive, then most malicious URLs will be detected. This is shown by the curves corresponding to $M_{0\text{-}8}(D_0)$. Also, if both the attacker and defender are adaptive and the attacker adapts its behavior a step further than the defender, parallel adaptation and full adaptation defense can still effectively detect the evasion, but the sequential adaptation defense failed to detect the evasion when the attacker manipulated two features. We can see this from curves corresponding to $M_{0\text{-}8}(D_9)$. True-positive rate of both parallel adaptation and full adaptation defense changes little, while true-positive rate of sequential adaptation defense drops to zero when two features are manipulated. A possible explanation is that D_9 in sequential adaptation strategy is manipulated from D_8, which is comparable to the scenario that D_1 is manipulated from D_0. This is also reflected on the two overlapping curves $M_0(D_1)$ and $M_{0\text{-}8}(D_9)$.

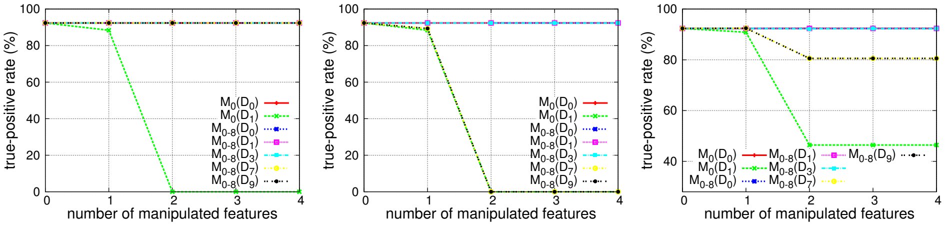


(a) Parallel adaptation (b) Sequential adaptation (c) Full adaptation

Figure 6: Comparison of cross-layer adaptive defense against adaptive attacks: data-level aggregation.

Figure 7 plots the true-positive rate in the case of AND-aggregation, which is similar to the results in the case of data-level aggregation. For example, if the attacker is adaptive but the defender is non-adaptive, most malicious URLs will not be detected because the true-positive rate of $M_0(D_1)$ becomes 0% when the attacker manipulates two features in the cases of parallel and sequential adaptations.

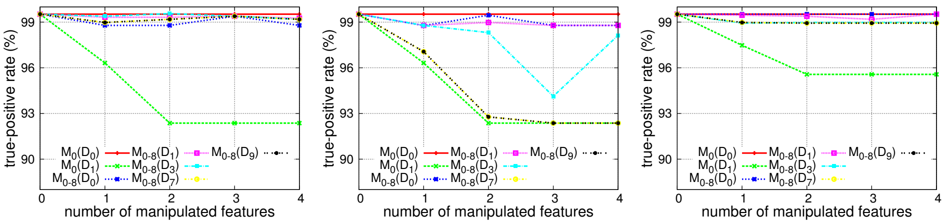
Figure 8 plots the results in the case of OR-aggregation detection. We observe that if the attacker is adaptive but the defender is non-adaptive, some malicious URLs will not be detected. This can be seen from the fact that the true-positive rate of $M_0(D_1)$ drops 4%-8% when the attacker adapts its behavior by manipulating two features. Note



(a) Parallel adaptation (b) Sequential adaptation (c) Full adaptation

Figure 7: Comparison of cross-layer adaptive defense against adaptive attacks: AND-aggregation.

that this is different from the data-level aggregation and AND-aggregation defense, where the true-positive rate drops to 0% when two features are manipulated. This can be explained by the fact that the OR-aggregation has naturally high true-positive. For the cases where attacker is non-adaptive and defender is adaptive, or both attacker and defender are adaptive, we observe the same phenomenon as the data-level and AND-aggregation defense. To summarize, OR-aggregation adaptive defense is more resilient than data-level aggregation and AND-aggregation adaptive defense.



(a) Parallel adaptation (b) Sequential adaptation (c) Full adaptation

Figure 8: Comparison of cross-layer adaptive defense against adaptive attacks: OR-aggregation.

Resilience measurement through false-positive rate:

As mentioned above, evaluation based on true-positive rate alone can be misleading. Table 4 describes the true-positive rate, false-negative rate, and false-positive rate of cross-layer adaptive defense against adaptive and non-adaptive attacks. We have the following observations.

First, if the attacker is not adaptive and the defender is adaptive, using more classifiers based on full-adaptation will achieve effective detection. This can be seen from columns $M_0(D_0)$ and $M_{0\text{-}8}(D_0)$. For parallel and sequential adaptation strategies, data-level aggregation and OR-aggregation detections $M_{0\text{-}8}(D_0)$ have higher false-negative rate than $M_0(D_0)$, and AND-aggregation adaptive detection using either $M_{0\text{-}8}(D_0)$ or $M_0(D_0)$ has the same true-positive rate. For full-adaptation strategy, using either $M_{0\text{-}8}(D_0)$ or $M_0(D_0)$ for cross-layer detection has the same false-positive rate, which shows the advantage of full-adaptation strategy.

Second, if both attackers and defenders are adaptive, using a larger number of adaptive classifiers can significantly lower both false-negative and false-positive rates. This can be seen from columns $M_0(D_1)$ and $M_{0\text{-}8}(D_1)$. For all adaptation strategies, using classifier $M_{0\text{-}8}$ in all three aggregation methods can achieve higher true-positive rate. For parallel and sequential adaptation strategies, data-level aggregation and AND-aggregation detections $M_0(D_1)$ failed to detect the malicious websites, but OR-aggregation detection can achieve 92.4% true-positive rate.

Third, OR-aggregation detection using full-adaptation strategy can achieve the best true-positive rate. We can see this from vertical perspective of Table 4. Especially, if the defender is adaptive, whether the attacker is adaptive or not, OR-aggregation can achieve a better true-positive rate.

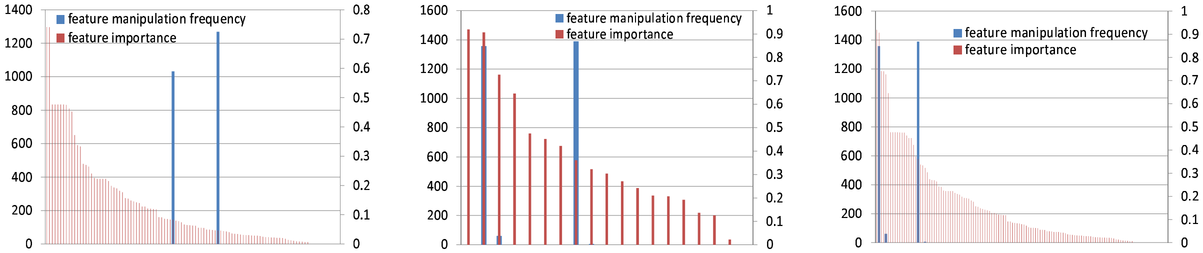
Are the manipulated features the important ones?

One may expect that the important features are the ones that will be manipulated by the attacker in attempting to evade defense. It is somewhat surprising to see that this is not necessarily the case.

Figure 9 shows which features are manipulated by the attacker so as to bypass classifier M_0. In order to evade the 1,467 malicious URLs from the defense, our algorithm manipulated a few features. We observe that there is no simple

Strategy	Cross-layer aggregation method	$M_0(D_0)$			$M_{0\text{-}8}(D_0)$			$M_0(D_1)$			$M_{0\text{-}8}(D_1)$		
		TP	FN	FP	TP	FN	FP	TP	FN	FP	TP	FN	FP
Parallel	data-level aggregation	99.52	0.48	0.03	98.70	1.30	0.02	0.00	100.00	0.03	99.11	0.89	0.02
	AND-aggregation	92.37	7.63	0.00	92.37	7.63	0.00	0.00	100.00	0.00	92.37	7.63	0.00
	OR-aggregation	99.52	0.48	0.06	98.70	1.30	0.05	92.37	7.63	0.06	99.32	0.68	0.05
Sequential	data-level aggregation	99.52	0.48	0.03	98.70	1.30	0.00	0.00	100.00	0.03	98.70	1.30	0.00
	AND-aggregation	92.37	7.63	0.00	92.37	7.63	0.00	0.00	100.00	0.00	92.37	7.63	0.00
	OR-aggregation	99.52	0.48	0.06	98.77	1.23	0.03	92.37	7.63	0.06	98.70	1.30	0.03
Full	data-level aggregation	99.52	0.48	0.03	99.52	0.48	0.01	49.63	50.37	0.03	99.52	0.48	0.01
	AND-aggregation	92.37	7.63	0.00	92.37	7.63	0.00	46.42	53.58	0.00	92.37	7.63	0.00
	OR-aggregation	99.52	0.48	0.06	99.52	0.48	0.05	95.57	4.43	0.06	99.52	0.48	0.05

Table 4: Adaptive defense vs. (non-)adaptive attack using cross-layer detection (TP: true-positive rate; FN: false-negative rate; FP: false-positive rate). Note that TP + FN =1. Four features are manipulated in all cases.



(a) Application-layer (b) Network-layer (c) `data`-level aggregation

Figure 9: Important features vs. manipulated features (red bar: feature importance according to `GainRatioAttributeEval`; blue bar: frequency a feature is manipulated when manipulating the 1,467 malicious URLs; the blue bars overlay some red bars).

correspondence between the most often manipulated features and the most important features, which were ranked using the `GainRatioAttributeEval` feature selection method mentioned in Section 2.3.

- At the application layer, which is relevant because of the `AND`- and `OR`-aggregation, two features, `Postal_code` and `Redirect_num`, need to be manipulated in order to evade application-layer classifier M_0. These two features are not very important in terms of their contributions to the classifiers, but their manipulation allows the attacker to evade detection. This phenomenon tells us that non-important features can also play an important role in evading detection. The reason that only two features need to be manipulated can be attributed to the fact that the application-layer decision tree is unbalanced and has short paths.
- At the network layer, which is relevant because of the `AND`- and `OR`-aggregation, four features are manipulated to evade M_0. The four features are: `Dist_remote_TCP_port`, `Dist_remote_IP`, `Duration`, `Local_app_packets`, which are ranked the 2nd, 3rd, 8th, 9th in terms of their importance to the classifier. However, they are most often manipulated because they correspond to nodes that are typically parents of the leaves that indicate malicious URLs. Another two features are important features.
- At the cross layer, there are only four features that need be manipulated in order to evade the detection of cross-layer M_0 as shown in Table 4. Like the network-layer defense, the manipulation of four features will lead to a high evasion success rate. The four features are the same as the network-layer ones that are manipulated.

The proceeding phenomenon, namely that the manipulated features are not necessarily the important features, can be explained as follows. Our manipulation algorithm often manipulates the features that are parents of the leaves, which are often less important than the features that are close to the root.

What kinds of decision-trees are more resilient?

From the defender's perspective, `OR`-aggregation cross-layer detection is better than data-level aggregation and `AND`-aggregation cross-layer detection, and full adaptation is better than parallel and sequential adaptations in the investigated scenarios. Perhaps more importantly, we observe that from the defender's perspective, less important features are also crucial to correct classification. If one wants to build a resilient decision tree, we offer the following guideline.

> A decision tree is more resilient against adaptive attacks if it is balanced and tall. This is because a short path will make it easy to evade the detection.

To justify these guidelines, we note that the heights of the application-, network- and data-level aggregation cross-layer decision trees are 4, 6 and 6, respectively. Our experiments showed that the attacker only needs to manipulate

	app-layer	net-layer	data-level aggregation
w/o feature selection	105/2	19/4	124/4
w/ feature selection	9/1	11/3	9/2

Table 5: # of manipulated features w/ or w/o feature selection (a/b: the input J48 classifier was learned from dataset of a features, of which b features are manipulated for evasion).

two features for evading the application-layer classifier, and to manipulate four features for evading both the network-layer and cross-layer classifiers. Moreover, let us look at Table 5, we observe the following. First, application-layer detection with feature selection is evaded by manipulating the `Whois_netType` feature, which is different from application-layer detection without feature selection. Second, network-layer detection with feature selection is evaded by manipulating three features (i.e., `Duration`, `Dist_TCP_port` and `App_packets`), where the former two features are the same as the network-layer detection without feature selection. Third, detection based on data-level aggregation is evaded by manipulating two features (i.e., `Dist_TCP_port` and `DNS_answer_times`), where the former is the same as the data-level aggregation detection without feature selection.

4 Related Work

Both industry and academia are actively seeking effective solutions to the problem of malicious websites. Industry has mainly offered their proprietary blacklists of malicious websites, such as Google's Safe Browsing [1]. Researchers have used Logistic regression to study phishing URLs [21], but without considering the issue of redirection. Redirection has been used as indicator of web spams [8, 37, 54, 44]. Kurt et al. [51] presented a system for scalably detecting spam contents. Ma et al. [31, 32] studied how to detect phishing and spams based on URLs themselves.

In terms of detecting malicious websites that may host malwares, Nazario developed PhoneyC to detect attacks that exploit *known* vulnerabilities [34], Choi et al. [13] investigated the detection of malicious URLs, and Canali et al. [11] presented the design and implementation of a static detection tool called Prophiler. However, all these studies did not consider the usefulness of cross-layer detection and adaptive attacks. On the other hand, the back-end system for deeper analysis is also an active research topic [15, 12, 55, 33], because attackers have been attempting to circumvent dynamic analysis [26, 46].

Our resilience study falls into the topic of *adversarial machine learning*, where the attacker aims to evade a detection mechanism that is derived from some machine learning method [7, 52]. In the context of unsupervised learning for anomaly detection, Perdisci et al. [41] investigated how to make the detection harder to evade. In the context of supervised learning, existing studies fall into two categories. In the first category, the attacker can poison/manipulate the training data. Existing studies for tackling this problem include [40, 36, 48]. In the second category, the training data is not poisoned. There are two scenarios. First, the attacker has black-box access to the detection mechanism and attempts to evade detection after making, ideally, as few as possible queries to the detection mechanism. This setting has been studied in [30, 35]. Second, the attacker has access to the detection mechanism, which is true for our resilience study. Dalvi et al. [18] used Game Theoretic method to study this problem in the setting of spam detection, which is reasonable because a rational spammer certainly would take into account the cost of attempting to evade detection. However, this is not necessarily applicable to our resilience study because the attacker does not have to be rational because any successful evasion will cause successful attacks. Moreover, our model actually gives the attacker more freedom since the attacker knows the data the defender collected.

5 Limitations and Future Work

Our method and system are not without limitations. First, our study is based on given URLs. For systematic detection and defense, it is ideal to crawl all domains and all URLs. Fortunately, it is relatively easy to extend our system from this perspective.

Second, our current data collection system may not be able to deal with obfuscated JavaScript-based redirection, which is a challenging open problem [20]. Although our collected data hints that JavaScript-based redirection is widely used by malicious websites, it appears that JavaScript obfuscation may not have been widely used because our system can effectively detect the malicious URLs. However, this is not true in general. It is perhaps equally important to deal with encrypted web contents. Fortunately, any progress in these directions can be plugged into our system in a modular fashion. We plan to enhance our static analysis system by incorporating new techniques such as those described in [15, 49, 39, 17].

Third, we considered a certain class of adaptive attacks that attempt to evade detection, where the same type of adaptation strategy is used by both attacker and defender. Our adaptive defense against these adaptive attacks was shown to be effective. This is based on the assumption that the defender knows the attacker's adaptation strategies, which may not be true in general. Moreover, there could be other classes of adaptive attacks that may be able to defeat our defense. Our artificial manipulation algorithms for mimicking adaptive attacks may or may not be truthful because when the algorithms changes the values of some application-layer features, some network-layer features may have to be changed correspondingly so as to keep the cross-layer semantics consistent. This is a non-trivial problem because there is no simple mapping between the application-layer features and the network-layer features. Finally, our adaptive defense is reminiscent of the studies in robust classifiers [10, 27, 22, 28, 9], which makes us believe that our study will spark more investigation on resilience, especially formal treatment and characterization on the power and limitation of our defense.

Fourth, our study is mainly based on the dataset we collected on a specific day. During our study, we also analyzed the datasets collected on several consecutive days, from which we obtained similar results. Nevertheless, we plan to conduct a systematic study over datasets that are collected by 90 consecutive days. We will report our results in a future version of the present paper.

6 Conclusion

We presented a novel approach for detecting malicious websites based on proactive static analysis of website contents. Unlike existing studies based on application-layer data alone, our cross-layer analysis takes advantage of the corresponding network-layer information. Moreover, we study the resilience of our solution against adaptive attacks. As discussed above, our study brings out several important and challenging research problems.

References

[1] Google safe browsing, `http://code.google.com/apis/safebrowsing/developers_guide_v2.html`.

[2] iptables 1.4.12.1 `www.netfilter.org/projects/iptables`.

[3] Know your enemy: Behind the scenes of malicious web servers. `http://www.honeynet.org`.

[4] Know your enemy: Malicious web servers. `http://www.honeynet.org`.

[5] Sophos corporation. security threat report update 07/2008. `http://sophos.com/sophos/docs/eng/papers/sophos-security-report-jul08-srna.pdf`.

[6] tcpdump 4.2.0 `www.tcpdump.org`.

[7] M. Barreno, B. Nelson, R. Sears, A. Joseph, and D. Tygar. Can machine learning be secure? In *ASIACCS'06*.

[8] A. Benczur, K. Csalogany, T. Sarlos, and M. Uher. Spamrank - fully automatic link spam detection. *AIRWeb'05*.

[9] B. Biggio, G. Fumera, and F. Roli. Multiple classifier systems under attack. In *MCS*, pages 74–83, 2010.

[10] L. Breiman. Bagging predictors. *Machine Learning*, pages 123–140, 1996.

[11] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. *WWW'11*.

[12] K. Chen, G. Gu, J. Nazario, X. Han, and J. Zhuge. WebPatrol: Automated collection and replay of web-based malware scenarios. In *ASIACCS'11*.

[13] H. Choi, B. Zhu, and H. Lee. Detecting malicious web links and identifying their attack types. In *WebApps'11*.

[14] C. Cortes and V. Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[15] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *WWW'10*.

[16] T. Cover and J. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[17] . Curtsinger, B. Livshits, B. Zorn, and C. Seifert. Zozzle: fast and precise in-browser javascript malware detection. In *USENIX Security*, 2011.

[18] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *KDD'04*.

[19] A. Dewald, T. Holz, and F. Freiling. Adsandbox: sandboxing javascript to fight malicious websites. In *SAC'10*.

[20] B. Feinstein and D. Peck. Caffeine monkey: Automated collection, detection and analysis of malicious javascript. In *DEFCON 15*, 2007.

[21] S. Garera, N. Provos, M. Chew, and A. Rubin. A framework for detection and measurement of phishing attacks. In *WORM'07*.

[22] A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *ICML'06*, 353–360.

[23] M. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato.

[24] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, pages 10–18, 2009.

[25] G. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. *UAI'95*, pages 338–345.

[26] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna. Escape from monkey island: Evading high-interaction honeyclients. In *DIMVA'11*.

[27] J. Kittler, M. Hatef, R. Duin, and J. Matas. On combining classifiers. *TPAMI'98*.

[28] A. Kolcz and C. Teo. Feature weighting for improved classifier robustness. In *CEAS*, 2009.

[29] S. Cessie and J. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 191–201, 1992.

[30] D. Lowd and C. Meek. Adversarial learning. In *KDD'05*, pages 641–647, 2005.

[31] J. Ma, L. Saul, S. Savage, and G. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *KDD'09*, pages 1245–1254.

[32] J. Ma, L. Saul, S. Savage, and G. Voelker. Identifying suspicious urls: an application of large-scale online learning. In *ICML'09*, pages 681–688, 2009.

[33] T. Frosch, M. Heiderich, and T. Holz. Iceshield: Detection and mitigation of malicious websites with a frozen dom. In *RAID'11*.

[34] J. Nazario. Phoneyc: A virtual client honeypot, 2009.

[35] B. Nelson, B. Rubinstein, L. Huang, A. Joseph, and D. Tygar. Classifier evasion: models and open problems. In *ECML PKDD 2011*, pages 92–98.

[36] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. *RAID'06*.

[37] Y. Niu, Y. Wang, H. Chen, M. Ma, and F. Hsu. A quantitative study of forum spamming using context-based analysis. In *NDSS*, 2007.

[38] T. Oates and D. Jensen. Large Datasets Lead to Overly Complex Models: An Explanation and a Solution. *KDD'98*.

[39] W. Palant. Javascript deobfuscator 1.5.8. `https://addons.mozilla.org/en-US/firefox/addon/javascript-deobfuscator/`, 2010.

[40] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proc. S&P'06*, 2006.

[41] R. Perdisci, G. Gu, and W.Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *ICDM'06*, pages 488–498.

[42] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*, pages 42–65. 1998.

[43] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose. All your iframes point to us. In *USENIX Security*, 2008.

[44] Z. Qian, Z. Mao, Y. Xie, and F. Yu. On network-level clusters for spam detection. In *NDSS'10*.

[45] R. Quinlan. *C4.5: Programs for Machine Learning*. San Mateo, CA, 1993.

[46] M. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt. Trends in circumventing web-malware detection. Technical report, Google, 2011.

[47] K. Rieck, T. Krueger, and A. Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *ACSAC'10*, pages 31–39, 2010.

[48] B. Rubinstein, B. Nelson, L. Huang, A. Joseph, S. Lau, S. Rao, N. Taft, and D. Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *IMC'09*, pages 1–14.

[49] Chenette S. `http://securitylabs.websense.com/content/Blogs/3198.aspx`.

[50] C. Seifert and R. Steenson. `https://projects.honeynet.org/capture-hpc`, 2006.

[51] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and Evaluation of a Real-Time URL Spam Filtering Service. In *Proc. IEEE S&P'11*, 2011.

[52] S. Venkataraman, A. Blum, and D. Song. Limits of learning-based signature generation with adversaries. *NDSS'08*.

[53] Y. Wang, D. Beck, X. Jiang, and R.Roussev. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *NDSS'06*, 2006.

[54] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *NDSS*, 2010.

[55] J. Zhang, C. Seifert, J. Stokes, and W. Lee. Arrow: Generating signatures to detect drive-by downloads. In *WWW*, pages 187–196, 2011.

Internet Mathematics Vol. 8, No. 3: 288–320

An Extended Stochastic Model for Quantitative Security Analysis of Networked Systems

Maochao Xu and Shouhuai Xu

Abstract. Quantitative security analysis of networked computer systems has been an open problem in computer security for decades. Recently, a promising approach was proposed in [Li et al. 11], which, however, made some strong assumptions including the exponential distribution of, and the independence among, the relevant random variables. In this paper, we substantially weaken these assumptions while offering, in addition to the same types of analytical results as in [Li et al. 11], methods for obtaining the desired security quantities in practice. Moreover, we investigate the problem from a higher-level abstraction, which also leads to both analytical results and practical methods for obtaining the desired security quantities. These should represent a significant step toward ultimately solving the problem of quantitative security analysis of networked computer systems.

1. Introduction

Quantitative security analysis of networked computer systems from a whole-system, rather than from an individual-component, perspective is one of the most important open problems in cyber security. Recently, a promising stochastic

ISSN: 1542-7951 print

process approach to tackling this problem was proposed in [Li et al. 11]. In this approach, a networked computer system is modeled as a complex network, or graph, $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, where V is the vertex, or node, set and E is the edge set over which direct attacks can be carried out. Then a stochastic abstraction of the interactions between the attacker and the defender taking place in G is considered. The main research task is to derive the following two security measures:

- $q(v)$: the probability that a node $v \in \mathsf{V}$ is compromised (or attacked) in the steady state.
- q: the probability that a uniformly chosen node in G is compromised (or attacked) in the steady state.

The quantity q is particularly important for decision-making because it captures or reflects the global security level of a networked system in a useful way. For example, given q, the defender can design or deploy appropriate cryptographic or security schemes to mitigate the damage of the attacks. (Some extensions to q will be discussed in Section 7.)

1.1. Our Contribution

In this paper, we make substantial improvements upon [Li et al. 11], which assumed that the relevant random variables follow the exponential distribution and are independent of each other. Specifically, we consider two cases: (I) the case that the lower-level attack–defense process is observed, which is the same as in [Li et al. 11] but with much weaker assumptions; (II) the case that the higher-level alternating renewal process, but not the lower-level attack–defense process, is observed. The latter case was not investigated in [Li et al. 11] and may be more realistic.

In the case that the lower-level attack–defense process is observed (Section 4), we present new analytical results for deriving $q(v)$ under weaker assumptions, where $q(v)$ is the probability that an arbitrary node $v \in \mathsf{V}$ is compromised in the steady state. On the one hand, these new analytical results lead to useful methods for obtaining the global security measure q in practice:

(i) In the case that G is a regular graph with moderate node degree and the dependence between the nodes can be ignored, we derive a closed-form solution for q.

(ii) In the case that G is an arbitrary graph (including the case of a regular graph but with large node degree) and the dependence between the nodes can be ignored, we give an approximation solution for q.

(iii) In the case that G is an arbitrary graph and the dependence between the nodes cannot be ignored, we describe a statistical solution for q.

On the other hand, the new analytical results lead to bounds of q both in the case that G is regular graph and in the case that G is an arbitrary graph. Such results are interesting and useful because they demand much less information. The upper bounds are especially useful in decision-making because they capture a sort of worst-case scenario from the defender's perspective.

In the case that the lower-level attack–defense process is not observed, but the higher-level alternating renewal process is observed (Section 5), we present new analytical results for estimating $q(v)$. These analytical results lead to statistical methods for obtaining the global security measure q in practice. Note that this case was not investigated in [Li et al. 11], and is harder to deal with analytically due to the lack of sufficient lower-level information.

Note that the results in [Li et al. 11] and in the present paper are applicable to both undirected and directed graphs. To simplify notation, we will focus on the case of undirected graphs, while noting that when we adapt the results to the case of directed graphs, we need to replace "node degree" with "node in-degree" (a node v's in-degree is the number of nodes that point to v).

1.2. Outline of the Paper

The rest of the paper is organized as follows. In Sections 2 and 3, we briefly review the model and results in [Li et al. 11] and some probabilistic notions, respectively. In Section 4, we present new results for the case in which the lower-level attack–defense process is observed. In Section 5, we present new results for the case in which the higher-level alternating renewal process is observed. In Section 6, we review related previous studies. In Section 7, we discuss the utility of the stochastic process approach and future research directions toward the ultimate goal. In Section 8, we conclude the paper.

1.3. Notation

The principal notation used throughout the paper is summarized as follows.

$\mathsf{G} = (\mathsf{V}, \mathsf{E})$**:** This is a graph that abstracts a networked system with $n = |\mathsf{V}|$ nodes of average node degree μ.

$\deg(v), q(v), K(v)$: In the steady state, a node $v \in \mathsf{V}$ is compromised with probability $q(v)$, and among the v's $\deg(v)$ neighbors, $K(v)$ neighbors are compromised.

q, K: In the steady state, the probability that a uniformly chosen node is compromised is denoted by q, and the number of a uniformly chosen node's compromised neighbors is denoted by K, where $q = \sum_{v \in \mathsf{V}} q(v)/n$ and $K = \sum_{v \in \mathsf{V}} K(v)/n$.

$\{(S_j(v), C_j(v)); j \geq 1\}$: This is the sequence of node v's cycles corresponding to the alternating renewal process, where each cycle consists of the time interval $S_j(v)$ during which v is secure and the time interval $C_j(v)$ during which v is compromised.

2. A Brief Review of the Model and Results in [Li et al. 11]

We start with a brief review of [Li et al. 11]. As mentioned above, a networked system is represented as a complex network $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, where V is the set of n nodes or vertices, namely $|\mathsf{V}| = n$, that abstract (for example) computers, and E is the set of edges that abstract the internode communications or interactions that can be exploited to carry out direct attacks. At any point in time, a node $v \in \mathsf{V}$ is either secure or compromised. These abstractions are not new. The novelty of [Li et al. 11] is the attack–defense process over G, which is specified by the following random variables:

- X_1: The time a secure node becomes compromised directly by the attacker outside of G. This models "drive-by download"-like attacks [Provos et al. 07].
- $X_{2,i}$: The time a secure node becomes compromised because of its ith compromised neighbor, $1 \leq i \leq \deg(v)$. This models attacks such as standard malware spreading within a network.
- Y_1: The time a compromised node becomes secure again because the compromise has been detected and cured. This models reactive defense.
- Y_2: The time a compromised node becomes secure again for any other reason (e.g., patching or reinstalling the software system even if the compromise was not detected). This models proactive defense.

As mentioned above, the main security measure of interest is q, the probability that a uniformly chosen node is compromised in the steady state. This immediately leads to the more intuitive security measure $q \cdot n$, namely the expected number of compromised nodes in the steady state. Since in the steady state an arbitrary node $v \in \mathsf{V}$ is compromised with probability $q(v)$, we have

$$q = \frac{\sum_{v \in \mathsf{V}} q(v)}{n}.$$

The above is perhaps the most important conceptual contribution of [Li et al. 11], which is very simple but insightful because q captures the global property of $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ in the presence of attacks and defenses. The main research task is to derive an expression for q, which turns out to be a difficult problem because the states of the nodes are random variables that are dependent on each other according to the complex structure of G. As a first step, [Li et al. 11] focused on the special case in which G is regular graph, namely all nodes have the same degree and the $q(v)$'s are all the same (i.e., independent of v). Based on the following assumption, two main results were offered in [Li et al. 11].

Assumption 2.1. [Li et al. 11]

1. Assumption on the distributions of the random variables:
   - X_1 follows the exponential distribution with rate α.
   - The $X_{2,i}$'s follow the exponential distribution with rate γ.
   - Y_1 follows the exponential distribution with rate β.
   - Y_2 follows the exponential distribution with rate η.
2. Assumption on the independence between the random variables: X_1, Y_1, Y_2, and the $X_{2,i}$'s are independent of each other.

Theorem 2.2. [Li et al. 11] *Suppose* G *is regular graph. Under Assumption 2.1, we have*

$$q = \frac{1}{1+m}, \quad \textit{where } m = \mathrm{E}\left[\frac{\beta+\eta}{\alpha+\gamma K}\right],$$

and the random variable K indicates the number of compromised neighbors of a node.

Theorem 2.3. [Li et al. 11] *For a regular graph* G *of node degree μ, under Assumption 2.1, we have*

$$\frac{\alpha}{\alpha+\beta+\eta} \leq q \leq \frac{\alpha+\gamma\mu}{\alpha+\beta+\eta+\gamma\mu}.$$

Simulation studies in [Li et al. 11] showed that the upper bound is reasonably tight not only for regular graphs, but also for Erdős–Rényi random graphs and power-law graphs (in the latter two cases, μ is naturally the average node degree). This means that we can plausibly use the upper bound in decision-making when there is a lack of information.

3. A Brief Review of Some Probabilistic Notions Used in This Paper

3.1. Some Probabilistic Distributions

A random variable Z is said to have an exponential distribution with rate $\lambda > 0$ if the distribution function is

$$F(z) = 1 - \exp\{-\lambda z\}.$$

This distribution has the memoryless property, which played an essential role in [Li et al. 11]. The exponential distribution is a special case of the Weibull distribution with distribution function

$$F(z) = 1 - \exp\{-(\gamma z)^{\alpha}\},$$

where $\gamma > 0$ and $\alpha > 0$ are scale and shape parameters, respectively.

The Lomax (or Pareto II) distribution has distribution function

$$F(z) = 1 - \left(1 + \frac{z}{\beta}\right)^{-\eta},$$

where $\beta > 0$ is the scale parameter and $\eta > 0$ is the shape parameter [Johnson et al. 94]. This distribution is interesting because it can model heavy-tailed phenomena.

3.2. Some Notions of Nonindependence between Random Variables

Two random variables Z_1 and Z_2 are said to be positively quadrant dependent (PQD) if for all $z_1, z_2 \in \mathbb{R}$,

$$P(Z_1 > z_1, Z_2 > z_2) \geq P(Z_1 > z_1)P(Z_2 > z_2).$$

In general, a family $\{Z_1, \ldots, Z_n\}$ of random variables is said to be positively upper orthant dependent (PUOD) if for all $z_i \in \mathbb{R}$, $i = 1, \ldots, n$,

$$P(Z_1 > z_1, \ldots, Z_n > z_n) \geq \prod_{i=1}^{n} P(Z_i > z_i).$$

A family $\{Z_1, \ldots, Z_n\}$ of random variables is said to be positively associated (PA) if

$$\mathrm{Cov}\left(f(Z_1, \ldots, Z_n),\, g(Z_1, \ldots, Z_n)\right) \geq 0$$

for all coordinatewise nondecreasing functions f, g on $\mathbb{R}^n$ as long as the covariance exists. Finally, we recall the multivariate Marshall–Olkin model, which offers a good deal of flexibility in modeling dependence structures [Marshall and Olkin 67, Li 08].

Let $\{E_B, B \subseteq \{1, 2, \ldots, b\}\}$ be a sequence of independent, exponentially distributed random variables, where E_B has mean $1/\lambda_B$. Let

$$T_j = \min\{E_B : j \in B\}, \quad j = 1, \ldots, b.$$

The joint distribution of $T = (T_1, \ldots, T_b)$ is called the Marshall–Olkin exponential distribution with parameters $\{\lambda_B, B \subseteq \{1, 2, \ldots, b\}\}$.

It is known [Barlow and Proschan 81, Theorem 3.2, Chapter 2] that PA $\Longrightarrow$ PUOD(PQD), meaning that PA is (not necessarily strictly) stronger than PUOD (PQD). It is also known that the Marshall–Olkin exponential distribution is PA, and hence PUOD [Mueller and Stoyan 02].

4. New Results for the Case in Which the Attack–Defense Process Is Observed

In this section, we present new analytical results for deriving $q(v)$, methods for deriving q in practice, and bounds of q in the absence of sufficient information.

4.1. New Analytical Results for Deriving $q(v)$ under Weaker Assumptions

We seek to substantially weaken Assumption 2.1 while considering more general graph structures (i.e., not just regular graphs). We start with the following assumption.

Assumption 4.1. (A substantially weaker assumption.)

1. Rather than assuming that X_1, the $X_{2,i}$'s, Y_1, and Y_2 are exponential random variables, we assume the following:
   - X_1 follows an arbitrary continuous distribution F_1 with finite mean.
   - $X_{2,i}$ follows an arbitrary continuous distribution $F_{2,i}$ with finite mean, where the distributions may be different for different i's.

- Y_1 follows an arbitrary continuous distribution G_1 with finite mean.
- Y_2 follows an arbitrary continuous distribution G_2 with finite mean.

2. Rather than assuming that X_1, the $X_{2,i}$'s, Y_1, and Y_2 are independent of each other, we assume that X_1, the $X_{2,i}$'s, Y_1, and Y_2 can be mutually dependent.

In practice, Assumption 4.1 can be tested using some nonparametric methods (cf. [Hollander and Douglas 99, Chapter 8]).

Theorem 4.2. *Denote the joint survival function for* $\{X_1, X_{2,1}, \ldots, X_{2,n}\}$ *by*

$$\bar{H}_{n+1}(x_1, \ldots, x_{n+1}) = P\left(X_1 > x_1, X_{2,1} > x_2, \ldots, X_{2,n} > x_{n+1}\right),$$

and the joint survival function for Y_1 *and* Y_2 *by*

$$\bar{G}(y_1, y_2) = P\left(Y_1 > y_1, Y_2 > y_2\right).$$

Under Assumption 4.1, the probability $q(v)$ *that an arbitrary node* $v \in \mathsf{V}$ *is* compromised *in the steady state is*

$$q(v) = \frac{1}{1+m}, \tag{4.1}$$

where

$$m = \mathrm{E}\left[\int_0^\infty \bar{H}_{K(v)+1}(x, \ldots, x)\, dx\right] \Big/ \int_0^\infty \bar{G}(x,x)\, dx,$$

and $K(v)$ *is the number of* compromised *neighbors of node* v.

Proof. Observe that in general, the states of node $v \in \mathsf{V}$ can be modeled as an alternating renewal process, with each cycle composed of two states: secure with random time $S_j(v)$ and compromised with random time $C_j(v)$ for $j = 1, 2, \ldots$. Under Assumption 4.1, for each cycle j we can express $(S_j(v), C_j(v))$ as

$$S_j(v) = S(v) = \min\left\{X_1, \{X_{2,i}; 1 \le i \le K(v)\}\right\}, \quad C_j(v) = C(v) = \min\left\{Y_1, Y_2\right\},$$

where $K(v)$ is the number of v's compromised neighbors. Hence,

$$\begin{aligned}
\mathrm{E}[S(v)] &= \mathrm{E}[\min\{X_1, X_{2,1}, \ldots, X_{2,K(v)}\}] \\
&= \mathrm{E}\left[\mathrm{E}\left[\min\{X_1, X_{2,1}, \ldots, X_{2,K(v)}\} \mid K(v)\right]\right] \\
&= \mathrm{E}\left[\int_0^\infty \bar{H}_{K(v)+1}(x, \ldots, x)\, dx\right].
\end{aligned}$$

Similarly, the time that a node is compromised in a cycle is

$$\mathrm{E}[C(v)] = \mathrm{E}[\min\{Y_1, Y_2\}] = \int_0^\infty \bar{G}(x,x)\, dx.$$

Note also that Assumption 4.1 implies that

$$\mathrm{E}[S(v) + C(v)] < \infty.$$

This and the fact that the distribution of $S(v) + C(v)$ is nonlattice guarantee (cf. [Ross 96, Theorem 3.4.4]) that the process is steady, and moreover,

$$q(v) = \frac{E[C(v)]}{E[S(v)] + E[C(v)]} = \frac{\int_0^\infty \bar{G}(x,x)\,dx}{\mathrm{E}\left[\int_0^\infty \bar{H}_{K(v)+1}(x,\ldots,x)\,dx\right] + \int_0^\infty \bar{G}(x,x)\,dx}.$$

The desired result follows. □

The following corollary relates Theorem 2.2 to Theorem 4.2.

Corollary 4.3. *Theorem 2.2 is a special case of Theorem 4.2.*

Proof. By replacing Assumption 4.1 with Assumption 2.1 and letting G be a regular graph in Theorem 4.2, we obtain

$$\begin{aligned}
m &= \mathrm{E}\left[\int_0^\infty \bar{H}_{K(v)+1}(x,\ldots,x)\,dx\,dx\right] \Big/ \int_0^\infty \bar{G}(x,x)\,dx \\
&= \mathrm{E}\left[\int_0^\infty \bar{F}_1(x) \prod_{i=1}^{K(v)} \bar{F}_{2,i}(x)\,dx\right] \Big/ \int_0^\infty \bar{G}_1(x)\bar{G}_2(x)\,dx \\
&= \mathrm{E}\left[\int_0^\infty \exp\{-\alpha x\}\exp\{-K(v)\cdot\gamma\}dx\right] \Big/ \int_0^\infty \exp\{-\beta x\}\exp\{-\eta x\}\,dx \\
&= \mathrm{E}\left[\frac{\beta+\eta}{\alpha+\gamma K(v)}\right].
\end{aligned}$$

Therefore, the resulting q, namely $q(v)$ in the case of regular graphs, is the same as in Theorem 2.2. □

Now we consider the case that the random variables are independent of each other but follow the Weibull distribution (rather than the exponential distribution as required in [Li et al. 11]).

Assumption 4.4. (Weaker than Assumption 2.1 but stronger than Assumption 4.1.)

1. Assumption on the distributions of the random variables:
   - X_1 follows the Weibull distribution with survival function $\bar{F}_1(t) = e^{-(\lambda_1 t)^\alpha}$.
   - The $X_{2,i}$'s follow the Weibull distribution with survival function $\bar{F}_2(t) = e^{-(\lambda_2 t)^\alpha}$.

- Y_1 follows the Weibull distribution with survival function $\bar{G}_1(t) = e^{-(\gamma_1 t)^\beta}$.
- Y_2 follows the Weibull distribution with survival function $\bar{G}_2(t) = e^{-(\gamma_2 t)^\beta}$.

2. X_1, the $X_{2,i}$'s, Y_1, and Y_2 are independent of each other (as in Assumption 2.1).

In practice, the Weibull distribution in Assumption 4.4 can be tested using standard statistical techniques such as QQ-plot and the Kolmogorov–Smirnov test (cf. [Lehmann and Romano 05, Chapter 14]). The following result is more general than Theorem 2.2 but less general than Theorem 4.2.

Theorem 4.5. *Replacing Assumption 4.1 in Theorem 4.2 with Assumption 4.4, we have*

$$m = \frac{\Gamma\left(1+\frac{1}{\alpha}\right)}{\Gamma\left(1+\frac{1}{\beta}\right)} \mathrm{E}\left[\frac{\left(\gamma_1^\beta + \gamma_2^\beta\right)^{1/\beta}}{\left(\lambda_1^\alpha + K(v)\cdot\lambda_2^\alpha\right)^{1/\alpha}}\right].$$

Proof. Note that

$$\begin{aligned}\int_0^\infty \bar{G}_1(x)\bar{G}_2(x)\,dx &= \int_0^\infty \exp\left\{-\left[(\gamma_1 x)^\beta + (\gamma_2 x)^\beta\right]\right\} dx \\ &= \int_0^\infty \exp\left\{-\left[\gamma_1^\beta + \gamma_2^\beta\right] x^\beta\right\} dx \\ &= \frac{1}{\left(\gamma_1^\beta + \gamma_2^\beta\right)^{1/\beta}} \Gamma\left(1+\frac{1}{\beta}\right),\end{aligned}$$

where $\Gamma(\cdot)$ is the gamma function. Note also that

$$\begin{aligned}\mathrm{E}\left[\int_0^\infty \bar{F}_1(x)\bar{F}_2^{K(v)}(x)\,dx\right] &= \mathrm{E}\left[\int_0^\infty \exp\{-[(\lambda_1 x)^\alpha + K(v)\cdot(\lambda_2 x)^\alpha]\}\,dx\right] \\ &= \mathrm{E}\left[\int_0^\infty \exp\{-[(\lambda_1^\alpha + K(v)\cdot\lambda_2^\alpha]\,x^\alpha\}\,dx\right] \\ &= \Gamma\left(1+\frac{1}{\alpha}\right)\mathrm{E}\left[\frac{1}{\lambda_1^\alpha + K(v)\cdot\lambda_2^\alpha}\right]^{1/\alpha}.\end{aligned}$$

Using Theorem 4.2, we have the desired result. □

4.2. Applying the Analytical Results to Obtain q in Practice

Now we show how to use the above analytical results to derive methods for obtaining q. While allowing X_1, the $X_{2,i}$'s, Y_1, and Y_2 to be possibly dependent on each other, we further consider the (in)dependence between the states of a node's neighbors. This leads to three cases:

(i) In the case that G is a regular graph of node degree μ and the dependence between neighbors can be ignored, we offer closed-form results in Section 4.2.1.

(ii) In the case that G is an arbitrary graph and the dependence between neighbors can be ignored, we offer approximation results in Section 4.2.2.

(iii) In the case that G is an arbitrary graph and the dependence between the neighbors cannot be ignored, we offer statistical results in Section 4.2.3.

4.2.1. Regular Graphs with Ignorable Dependence between Nodes. In case (i), every node has μ neighbors, denoted by $1, 2, \ldots, \mu$, which are compromised in the steady state with probability $q_1 = q_2 = \cdots = q_\mu = q$. Let I_i be a Bernoulli random variable with parameter q, which is the probability that the ith neighbor is compromised. Then $K(v) = \sum_{i=1}^{\mu} I_i$ is a binomial distribution with parameter q, and

$$P(K(v) = k) = \binom{\mu}{k} q^k (1-q)^{\mu-k}, \quad k = 0, \ldots, \mu.$$

According to Theorem 4.2, which requires Assumption 4.1, we have

$$\begin{aligned} m &= \mathrm{E}\left[\int_0^\infty \bar{H}_{K(v)+1}(x, \ldots, x)\, dx\right] \Big/ \int_0^\infty \bar{G}(x,x)\, dx \\ &= \sum_{k=0}^{\mu} \left[\binom{\mu}{k} q^k (1-q)^{\mu-k} \int_0^\infty \bar{H}_{k+1}(x, \ldots, x)\, dx\right] \Big/ \int_0^\infty \bar{G}(x,x)\, dx. \end{aligned}$$

Hence, we have

$$q + \sum_{k=0}^{\mu} \left[\binom{\mu}{k} q^{k+1} (1-q)^{\mu-k} \int_0^\infty \bar{H}_{k+1}(x, \ldots, x) dx\right] \Big/ \int_0^\infty \bar{G}(x,x)\, dx = 1. \tag{4.2}$$

In order to solve (4.2), we need to know the specific $\bar{H}_{k+1}(x, \ldots, x)$ and $\bar{G}(x,x)$ that can be obtained in scenarios such as the following:

- Closed-form result under Assumption 2.1, which is stronger than Assumption 4.1: In this special case, we have Theorem 2.2 and thus (4.2) becomes

$$q + \sum_{k=0}^{\mu} \frac{\beta+\eta}{\alpha+\gamma k}\binom{\mu}{k} q^{k+1}(1-q)^{\mu-k} = 1. \tag{4.3}$$

- Closed-form result under Assumption 4.4, which is stronger than Assumption 4.1. In this special case, we have Theorem 4.5, and thus (4.2) becomes

$$q + \sum_{k=0}^{\mu} \binom{\mu}{k} q^{k+1}(1-q)^{\mu-k} \frac{\Gamma\left(1+\frac{1}{\alpha}\right)}{\Gamma\left(1+\frac{1}{\beta}\right)} \left[\frac{\left(\gamma_1^{\beta}+\gamma_2^{\beta}\right)^{1/\beta}}{\left(\lambda_1^{\alpha}+k\lambda_2^{\alpha}\right)^{1/\alpha}}\right] = 1. \tag{4.4}$$

When μ is not large, the solutions for q in (4.3) and (4.4) can be derived using an appropriate iterative algorithm, such as the Newton–Raphson method [Stoer and Bulirsch 02]. When μ is large, it might be infeasible to derive explicit solutions for q, but we can instead use the approximation result in Section 4.2.2.

4.2.2. Arbitrary Graphs with Ignorable Dependence between the Nodes.

For an arbitrary node $v \in \mathsf{V}$, let I_i be a Bernoulli random variable with parameter q_i, which is the probability that the ith neighbor of node v is compromised. Note that $K(v) = \sum_{i=1}^{\deg(v)} I_i$. Suppose the mean and variance of $K(v)$ exist and we can estimate them (e.g., based on sufficiently many observations on v's neighbors). Set

$$\bar{k} = \mathrm{E}[K(v)], \quad \sigma^2 = \mathrm{Var}(K(v)).$$

In this case, we are unable to give closed-form solutions, and instead propose using two approximations:

(a) Normal approximation: If $\sigma^2 \to \infty$ as $\deg(v) \to \infty$, by Lindeberg's theorem (cf. [Billingsley 95, p. 359]),

$$P\left(K(v) \le x\right) \approx \int_0^x \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(y-\bar{k})^2}{2\sigma^2}\right\} dy.$$

According to Theorem 4.2, which requires Assumption 4.1, we have

$$\begin{aligned} m &= \mathrm{E}\left[\int_0^\infty \bar{H}_{K(v)+1}(x,\ldots,x)\,dx\right] \Big/ \int_0^\infty \bar{G}(x,x)\,dx \\ &\approx \int_0^\infty \int_0^{\deg(v)} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(k-\bar{k})^2}{2\sigma^2}\right\} \\ &\quad \times \bar{H}_{k+1}(x,\ldots,x)\,dk\,dx \Big/ \int_0^\infty \bar{G}(x,x)\,dx. \end{aligned} \tag{4.5}$$

As a result, we can obtain more specific approximation solutions as follows:

- Approximation result under Assumption 2.1, which is stronger than Assumption 4.1. In this special case we have Theorem 2.2, and thus (4.5) becomes

$$m \approx \frac{\beta+\eta}{\sqrt{2\pi}\sigma}\int_0^{\deg(v)} \frac{1}{\alpha+\gamma x}\exp\left\{-\frac{(k-\bar{k})^2}{2\sigma^2}\right\} dk.$$

- Approximation result under Assumption 4.4, which is stronger than Assumption 4.1. In this special case, we have Theorem 4.5, and thus (4.5) becomes

$$\begin{aligned} m \approx{} & \frac{\Gamma\left(1+\frac{1}{\alpha}\right)\left(\gamma_1^\beta+\gamma_2^\beta\right)^{1/\beta}}{\sqrt{2\pi}\sigma\Gamma\left(1+\frac{1}{\beta}\right)} \\ & \times\int_0^{\deg(v)} \frac{1}{\left(\lambda_1^\alpha+k\lambda_2^\alpha\right)^{1/\alpha}}\exp\left\{-\frac{(k-\bar{k})^2}{2\sigma^2}\right\} dk. \end{aligned}$$

(b) Poisson approximation. Note that $\bar{k}\geq 0$. If $\deg(v)\to\infty$ and

$$\max\{q_1,\ldots,q_{\deg(v)}\}\to 0,$$

then [Billingsley 95, Theorem 23.2] says that

$$P(K(v)=i)\approx \exp\left\{-\bar{k}\right\}\frac{(\bar{k})^i}{i!},\quad i=0,1,\ldots,\deg(v).$$

According to Theorem 4.2, which requires Assumption 4.1, we have

$$\begin{aligned} m &= \mathrm{E}\left[\int_0^\infty \bar{H}_{K(v)+1}(x,\ldots,x)\,dx\right]\Big/\int_0^\infty \bar{G}(x,x)\,dx \\ &\approx \sum_{i=0}^{\deg(v)}\exp\left\{-\bar{k}\right\}\frac{(\bar{k})^i}{i!}\int_0^\infty \bar{H}_{i+1}(x,\ldots,x)\,dx\Big/\int_0^\infty \bar{G}(x,x)\,dx. \end{aligned} \tag{4.6}$$

As a result, we can have more specific approximation solutions as follows:

- Approximation result under Assumption 2.1, which is stronger than Assumption 4.1. In this special case, we have Theorem 2.2, and thus (4.6) becomes

$$m\approx(\beta+\eta)\exp\left\{-\bar{k}\right\}\sum_{i=0}^{\deg(v)}\frac{1}{\alpha+\gamma i}\frac{(\bar{k})^i}{i!}.$$

- Approximation result under Assumption 4.4, which is stronger than Assumption 4.1. In this special case, we have Theorem 4.5, and thus

(4.6) becomes

$$m \approx \frac{\Gamma\left(1+\frac{1}{\alpha}\right)\left(\gamma_1^{\beta}+\gamma_2^{\beta}\right)^{1/\beta}}{\Gamma\left(1+\frac{1}{\beta}\right)} \sum_{i=0}^{\deg(v)} \frac{(\bar{k})^i \exp\left\{-\bar{k}\right\}}{\left(\lambda_1^{\alpha}+i\cdot\lambda_2^{\alpha}\right)^{1/\alpha} i!}.$$

In both normal approximation and Poisson approximation, having obtained m allows one to derive $q(v)$ according to (4.1) in Theorem 4.2. Note that this method is sound because both Assumption 2.1 and Assumption 4.4 are stronger than Assumption 4.1, which underlies Theorem 4.2. From a sufficiently large sample $\mathsf{V}' \subset \mathsf{V}$ with each $v \in \mathsf{V}'$ being uniformly chosen from V, we can derive $q = \sum_{v\in\mathsf{V}'} q(v)/|\mathsf{V}'|$.

4.2.3. Arbitrary Graphs with Nonignorable Dependence between the Nodes. If the dependence between neighbors cannot be ignored, the probability mass function of $K(v)$ may be estimated from data as follows. Suppose we have observations $(k_1, k_2, \ldots, k_b)$ on $K(v)$, where b is the number of observations on the number of v's compromised neighbors in the steady state (i.e., at b different points of observation time). Then the probability mass function of $K(v)$ can be estimated as

$$p_k = P(K(v)=k) = \frac{\sum_{i=1}^{b} \mathrm{I}(k_i = k)}{b}, \quad b \to \infty,$$

where $\mathrm{I}(\cdot)$ is the indicator function. Thus, according to Theorem 4.2, we have

$$\begin{aligned} m &= \mathrm{E}\left[\int_0^{\infty} \bar{H}_{K(v)+1}(x,\ldots,x)\,dx\right] \Big/ \int_0^{\infty} \bar{G}(x,x)\,dx \\ &= \sum_{k=0}^{\deg(v)} p_k \int_0^{\infty} \bar{H}_{k+1}(x,\ldots,x)\,dx \Big/ \int_0^{\infty} \bar{G}(x,x)\,dx. \qquad (4.7) \end{aligned}$$

To be specific, let us consider a concrete example under the following assumption:

Assumption 4.6. (Weaker than Assumption 2.1 but stronger than Assumption 4.1.)

1. Assumption on the distributions of the random variables:
   - X_1 follows the exponential distribution with parameter λ.
   - The $X_{2,i}$'s follow the multivariate Marshall–Olkin exponential model.
   - Y_1 and Y_2 follow the bivariate Marshall–Olkin exponential model.
2. The $X_{2,i}$'s can be dependent on each other; Y_1 and Y_2 can be dependent on each other.

Under Assumption 4.6, the joint survival function of $(X_{2,1},\dots,X_{2,k})$ is

$$\begin{aligned}\bar{H}_k(x_1,\dots,x_k) &= P(X_{2,1} > x_1,\dots,X_{2,k} > x_k)\\ &= \exp\Bigg\{-\sum_{i=1}^{k}\lambda_i x_i - \sum_{i<j}\lambda_{ij}\max\{x_i,x_j\} - \lambda_{12\dots k}\max\{x_1,\dots,x_k\}\Bigg\}.\end{aligned}$$

Therefore,

$$\begin{aligned}\int_0^\infty \bar{H}_{k+1}(x,\dots,x)dx &= \int_0^\infty \exp\{-\lambda t\}\exp\{-\phi(\boldsymbol{\lambda},kt\}\,dt\\ &= \frac{1}{\lambda+\phi(\boldsymbol{\lambda},k)},\end{aligned}$$

where

$$\phi(\boldsymbol{\lambda},k) = \sum_{i=1}^{k}\lambda_i + \sum_{i<j}\lambda_{ij} + \cdots + \lambda_{12\dots k}.$$

Similarly, the joint survival function of (Y_1,Y_2) is

$$\begin{aligned}\bar{G}(y_1,y_2) &= P\left(Y_1 > y_1, Y_2 > y_2\right)\\ &= \exp\left\{-\gamma_1 y_1 - \gamma_2 y_2 - \gamma_{12}\max\{y_1,y_2\}\right\}.\end{aligned}$$

Then

$$\int_0^\infty \bar{G}(x,x)\,dx = \frac{1}{\gamma_1+\gamma_2+\gamma_{12}}.$$

Using (4.7), we have

$$m = (\gamma_1+\gamma_2+\gamma_{12})\sum_{k=0}^{\deg(v)}\frac{p_k}{\lambda+\phi(\boldsymbol{\lambda},k)}.$$

Having obtained m, one can derive $q(v)$ according to (4.1) in Theorem 4.2. Note that this method is sound because Assumption 4.6 is stronger than Assumption 4.1, which underlies Theorem 4.2. From a sufficiently large sample $\mathsf{V}' \subset \mathsf{V}$ with each $v \in \mathsf{V}'$ being uniformly chosen from V, we can derive $q = \sum_{v\in\mathsf{V}'} q(v)/|\mathsf{V}'|$.

4.3. New Bounds on q

Above, we discussed how to obtain q in various settings. When one cannot obtain q (for example, due to the lack of sufficient information), one can instead use the bounds of q in decision-making. This is plausible as long as the bounds are reasonably tight, and reasonable because it requires much less information to compute the bounds. In particular, one could use the upper bound of q in decision-making because it can be seen as a sort of worst-case scenario. In what follows we will differentiate the case of regular graphs from the case of arbitrary graphs under the following three assumptions:

Assumption 4.7. (Assumption underlying our general bounds of q.)

1. Assumption on the distributions of the random variables:
   - X_1 follows an arbitrary continuous distribution with survival function $\bar{F}_1$ and finite mean.
   - The $X_{2,i}$'s are PUOD with the same marginal continuous survival function $\bar{F}_2$ and finite mean.
   - Y_1 follows an arbitrary distribution with continuous survival function $\bar{G}_1$ and finite mean.
   - Y_2 follows an arbitrary distribution with continuous survival function $\bar{G}_2$ and finite mean.
2. X_1 and the $X_{2,i}$'s are independent, and Y_1 and Y_2 may be dependent on each other.

Assumption 4.8. (Assumption underlying one specific bound of q.)

1. Assumption on the distributions of the random variables:
   - X_1 follows the exponential distribution with rate α.
   - The $X_{2,i}$'s are PUOD with the same marginal distribution function F_2 of the exponential distribution with rate γ.
   - Y_1 follows the exponential distribution with rate β.
   - Y_2 follows the exponential distribution with rate η.
2. X_1 and the $X_{2,i}$'s are independent, and Y_1 and Y_2 are independent of each other.

Assumption 4.9. (Assumption underlying another specific bound of q.)

1. Assumption on the distributions of the random variables:
   - X_1 follows the Lomax distribution with scale parameter λ and shape parameter $\alpha_1 > 1$.
   - The $X_{2,i}$'s are PUOD with the same marginal distribution function F_2 of the Lomax distribution with scale parameter λ and shape parameter $\alpha_2 > 1$.

- Y_1 follows the Lomax distribution with scale parameter γ and shape parameter $\beta_1 > 1$.
- Y_2 follows the Lomax distribution with scale parameter γ and shape parameter $\beta_2 > 1$.

2. X_1 and the $X_{2,i}$'s are independent, and Y_1 and Y_2 are independent of each other.

In practice, the PUOD dependence assumption can be tested by the method suggested in [Scaillet 05], while the exponential distribution assumption and the Lomax distribution assumption can be tested using QQ-plot or the Kolmogorov–Smirnov test (see [Lehmann and Romano 05]).

4.3.1. Bounds on q in the Case of Regular Graphs.

We have the following theorem.

Theorem 4.10. *Suppose* G *is regular graph of node degree* μ. *Under Assumption 4.7, in the steady state we have*

$$\int_0^\infty \bar{G}(x,x)\,dx \Big/ \int_0^\infty \left[\bar{F}_1(x) + \bar{G}(x,x)\right]\,dx$$
$$\leq q \leq \int_0^\infty \bar{G}(x,x)dx \Big/ \int_0^\infty \left[\bar{F}_1(x)\bar{F}_2^{\bar{k}}(x) + \bar{G}(x,x)\right]\,dx,$$

where $\bar{k}$ *is the expected number of* compromised *neighbors for an arbitrary node.*

Proof. Denote by $1, 2, \ldots, \mu$ the neighbors of an arbitrary node $v \in \mathsf{V}$. Let I_i be the indicator function that $I_i = 1$ if and only if the ith neighbor of v is compromised, and let q_i be the probability that the ith neighbor is compromised. Under Assumption 4.7, the system will enter the steady state (see [Ross 96, Theorem 3.4.4]). Since G is regular graph, we have $q_1 = \cdots = q_\mu = q$. In the steady state, we have

$$\bar{k} = \mathrm{E}[K(v)] = \mathrm{E}\left[\sum_{i=1}^{\mu} I_i\right] = q\mu.$$

Assume that the $X_{2,i}$'s are PUOD with the same marginal survival function $\bar{F}_2$. Then

$$\begin{aligned} m &= \mathrm{E}\left[\int_0^\infty \bar{F}_1(x)\bar{H}_{K(v)}(x,\ldots,x)\,dx\right]\Big/\int_0^\infty \bar{G}(x,x)\,dx \\ &\geq \mathrm{E}\left[\int_0^\infty \bar{F}_1(x)\bar{F}_2^{K(v)}(x)\,dx\right]\Big/\int_0^\infty \bar{G}(x,x)\,dx \\ &\geq \left[\int_0^\infty \bar{F}_1(x)\bar{F}_2^{\mathrm{E}[K(v)]}(x)\,dx\right]\Big/\int_0^\infty \bar{G}(x,x)\,dx \\ &= \left[\int_0^\infty \bar{F}_1(x)\bar{F}_2^{\bar{k}}(x)dx\right]\Big/\int_0^\infty \bar{G}(x,x)\,dx, \end{aligned} \tag{4.8}$$

where the last inequality follows from Jensen's inequality. Using (4.1) in Theorem 4.2, one can obtain the upper bound of q.

On the other hand, we observe that for every $x \geq 0$, we have

$$\bar{H}_{K(v)}(x,\ldots,x)dx \leq 1.$$

The upper bound of m can be derived as

$$m \leq \int_0^\infty \bar{F}_1(x)\,dx\Big/\int_0^\infty \bar{G}(x,x)\,dx. \tag{4.9}$$

Using (4.1) in Theorem 4.2, one can obtain the lower bound of q. □

Theorem 4.11. *Suppose* G *is a regular graph of node degree μ. Under Assumption 4.8, in the steady state we have*

$$\frac{\alpha}{\alpha+\beta+\gamma} \leq q \leq \frac{-(\beta+\eta+\alpha-\mu\gamma)+\sqrt{(\beta+\eta+\alpha-\mu\gamma)^2+4\mu\gamma\alpha}}{2\mu\gamma}.$$

Proof. From (4.8), it follows that

$$m \geq \frac{\beta+\eta}{\alpha+q\mu\gamma}.$$

Using (4.1) in Theorem 4.2, we have

$$q = \frac{1}{m+1} \leq \frac{1}{\frac{\beta+\eta}{\alpha+q\mu\gamma}+1}.$$

This leads to

$$q^2\mu\gamma + q(\beta+\eta+\alpha-\mu\gamma) - \alpha \leq 0.$$

Hence

$$q \leq \frac{-(\beta+\eta+\alpha-\mu\gamma)+\sqrt{(\beta+\eta+\alpha-\mu\gamma)^2+4\mu\gamma\alpha}}{2\mu\gamma}.$$

On the other hand, according to (4.9) and (4.1), one can derive

$$q \geq \frac{\alpha}{\alpha + \beta + \gamma}.$$

□

Theorem 4.12. *Suppose* G *is a regular graph of node degree μ. Under Assumption 4.9, in the steady state we have*

$$\frac{\gamma(\alpha_1 - 1)}{\lambda(\beta_1 + \beta_2 - 1) + \gamma(\alpha_1 - 1)} \leq q \leq \frac{-B + \sqrt{B^2 + 4\gamma^2\alpha_2(\alpha_1 - 1)\mu}}{2\alpha_2\gamma\mu},$$

where $B = \lambda(\beta_1 + \beta_2 - 1) + \gamma(\alpha_1 - 1) - \gamma\alpha_2\mu$.

Proof. Note that F_i has Lomax survival function

$$\bar{F}_i(x) = \left(1 + \frac{x}{\lambda}\right)^{-\alpha_i}, \quad \alpha_i \geq 1, \quad i = 1, 2.$$

Similarly, G_i has Lomax survival function

$$\bar{G}_i(x) = \left(1 + \frac{x}{\gamma}\right)^{-\beta_i}, \quad \beta_i \geq 1, \quad i = 1, 2.$$

From (4.8), it follows that

$$m \geq \frac{\lambda}{\alpha_1 + \alpha_2\mu q - 1} \Big/ \frac{\gamma}{\beta_1 + \beta_2 - 1} = \frac{\lambda(\beta_1 + \beta_2 - 1)}{\gamma(\alpha_1 + \alpha_2\mu q - 1)}.$$

Using (4.1) in Theorem 4.2, we have

$$q \leq \frac{\gamma(\alpha_1 + \alpha_2\mu q - 1)}{\lambda(\beta_1 + \beta_2 - 1) + \gamma(\alpha_1 + \alpha_2\mu q - 1)}.$$

This leads to

$$q^2\alpha_2\gamma\mu + qB - \gamma(\alpha_1 - 1) \leq 0,$$

where

$$B = \lambda(\beta_1 + \beta_2 - 1) + \gamma(\alpha_1 - 1) - \gamma\alpha_2\mu.$$

Hence, the upper bound of q is

$$q \leq \frac{-B + \sqrt{B^2 + 4\gamma^2\alpha_2(\alpha_1 - 1)\mu}}{2\alpha_2\gamma\mu}.$$

On the other hand, according to (4.9) and (4.1), we have

$$m \leq \frac{\lambda(\beta_1 + \beta_2 - 1)}{\gamma(\alpha_1 - 1)}.$$

So the lower bound of q is

$$q \geq \frac{\gamma(\alpha_1 - 1)}{\lambda(\beta_1 + \beta_2 - 1) + \gamma(\alpha_1 - 1)}.$$

□

4.3.2. Bounds of q in the Case of Arbitrary Graphs. For an arbitrary node $v \in \mathsf{V}$, it follows that

$$K(v) = \sum_{i=1}^{\deg(v)} I_i,$$

where I_i is a Bernoulli random variable with parameter q_i, which is the probability that the ith neighbor of v is compromised. Note that the I_i's may be dependent on each other and the q_i's may be different. Note also that in this case, we cannot obtain a counterpart of Theorem 4.10, because the concavity needed to apply Jensen's inequality may not hold. As such, we are able to give only the following specific results in this more general case.

Theorem 4.13. *Suppose* G *is an arbitrary graph with average node degree* μ*. Under Assumption 4.8, in the steady state we have*

$$\frac{\alpha}{\alpha + \beta + \gamma} \leq q \leq \frac{\alpha + \gamma\mu}{\alpha + \beta + \eta + \gamma\mu}.$$

Proof. In a fashion similar to the proof of Theorem 4.10, we can show that for an arbitrary node v,

$$\frac{\alpha}{\alpha + \beta + \gamma} \leq q(v) \leq \frac{\alpha + \gamma\,\mathrm{E}[K(v)]}{\alpha + \beta + \eta + \gamma\,\mathrm{E}[K(v)]}.$$

Since

$$q = \frac{\sum_{v \in \mathsf{V}} q(v)}{n},$$

it follows that

$$q \geq \frac{\alpha}{\alpha + \beta + \gamma}.$$

On the other hand, because

$$\mathrm{E}[K(v)] = \mathrm{E}\left[\sum_{i=1}^{\deg(v)} I_i\right] = \sum_{i=1}^{\deg(v)} q_i \leq \deg(v),$$

we have

$$\frac{\alpha + \gamma \operatorname{E}[K(v)]}{\alpha + \beta + \eta + \gamma \operatorname{E}[K(v)]} \leq \frac{\alpha + \gamma \deg(v)}{\alpha + \beta + \eta + \gamma \deg(v)}.$$

Hence

$$q(v) \leq \frac{\alpha + \gamma \deg(v)}{\alpha + \beta + \eta + \gamma \deg(v)}.$$

Therefore,

$$q \leq \frac{1}{n} \sum_{v \in \mathsf{V}} \frac{\alpha + \gamma \deg(v)}{\alpha + \beta + \eta + \gamma \deg(v)}. \tag{4.10}$$

Because

$$h(x) = \frac{\alpha + \gamma x}{\alpha + \beta + \eta + \gamma x}, \quad x \geq 0,$$

is a concave function, Jensen's inequality yields

$$\frac{1}{n} \sum_{v \in \mathsf{V}} \frac{\alpha + \gamma \deg(v)}{\alpha + \beta + \eta + \gamma \deg(v)} \leq \frac{\alpha + \gamma \frac{1}{n} \sum_{v \in \mathsf{V}} \deg(v)}{\alpha + \beta + \eta + \gamma \frac{1}{n} \sum_{v \in \mathsf{V}} \deg(v)}.$$

Hence, by (4.10), we have

$$q \leq \frac{\alpha + \gamma \mu}{\alpha + \beta + \eta + \gamma \mu}.$$

This completes the proof. □

In a similar fashion, we can prove the following result.

Theorem 4.14. *Suppose* G *is an arbitrary graph with average degree* μ. *Under Assumption 4.9, in the steady state we have*

$$\frac{\gamma(\alpha_1 - 1)}{\lambda(\beta_1 + \beta_2 - 1) + \gamma(\alpha_1 - 1)} \leq q \leq \frac{\gamma(\alpha_1 + \alpha_2 \mu - 1)}{\lambda(\beta_1 + \beta_2 - 1) + \gamma(\alpha_1 + \alpha_2 \mu - 1)}.$$

5. New Results for the Case in Which Only the Alternating Renewal Process Is Observed

The results obtained thus far are applicable when the lower-level attack–defense process is observed, or more specifically the distributions F_1, the $F_{2,i}$'s, G_1, and G_2 and their dependence structure (if applicable) are observed. What if they are not observed? This is a very reasonable question, because obtaining all detailed information can be very costly, and it may be much simpler to observe the higher-level alternating renewal process $\{(S_j(v), C_j(v)); j \geq 1\}$ mentioned above. In this case, neither the results in [Li et al. 11] nor the results described above are applicable, which motivates us to present the following results for this possibly more realistic scenario.

5.1. Analytical Results for Estimating $q(v)$

Let $p_v(t)$ be the probability that node v is secure at time t, and let the limiting average probability be

$$\bar{p}(v) = \lim_{t\to\infty} \frac{1}{t} \int_0^t p_v(s)\, ds.$$

For node $v \in \mathsf{V}$ with $\{(S_j(v), C_j(v)) : 1 \leq j \leq b\}$, define

$$W_b(v) = S_1(v) + S_2(v) + \cdots + S_b(v), \qquad D_b(v) = C_1(v) + C_2(v) + \cdots + C_b(v).$$

Lemma 5.1. [Marlow and Tortorella 95, Theorem 1] *If*

$$\lim_{b\to\infty} \frac{W_b(v)}{b} = w(v) \ \ a.s. \quad \text{and} \quad \lim_{b\to\infty} \frac{D_b(v)}{b} = d(v) \ \ a.s.,$$

where "a.s." means "almost surely" and $0 < w(v), d(v) < \infty$*, then*

$$\bar{p}(v) = \frac{w(v)}{w(v) + d(v)}.$$

Assumption 5.2. Assume that $\{S_j(v) : j \geq 1\}$ and $\{C_j(v) : j \geq 1\}$ are sequences of pairwise PQD random variables with finite variances, and

(a) $\sum_{j=1}^{\infty} j^{-2}\mathrm{Cov}\,(S_j(v), W_j(v)) < \infty$;

(b) $\sum_{j=1}^{\infty} j^{-2}\mathrm{Cov}\,(C_j(v), D_j(v)) < \infty$;

(c) $\sup_{j\geq 1} \mathrm{E}\,|S_j(v) - \mathrm{E}[S_j(v)]| < \infty$;

(d) $\sup_{j\geq 1} \mathrm{E}\,|C_j(v) - \mathrm{E}[C_j(v)]| < \infty$.

In practice, the above assumption can be tested as follows: PQD dependence can be tested using the statistical methods described in [Denuit and Scaillet 04]. To test for finite variance, we may use the methods described in [Resnick 07] for heavy-tailed data. If the data are not extremely heavy-tailed, we may assume that the data have finite variance.

Lemma 5.3. [Birkel 89, Theorem 1] *Let $\{S_j(v) : j \geq 1\}$ be a sequence of pairwise PQD random variables with finite variance. Assume*

(a) $\sum_{j=1}^{\infty} j^{-2}\mathrm{Cov}\,(S_j(v), W_j(v)) < \infty$.

(b) $\sup_{j\geq 1} \mathrm{E}\,|S_j(v) - \mathrm{E}[S_j(v)]| < \infty$.

Then as $b \to \infty$, we have $b^{-1}(W_b(v) - \mathrm{E}[W_b(v)]) \to 0$ almost surely.

Theorem 5.4. *Under Assumption 5.2 and given a sufficiently large number b of observations on the states of node v, the probability $q(v)$, namely the probability that node v is* compromised*, if it exists (i.e., the alternating renewal process is steady), can be expressed as*

$$q(v) = \frac{d^*(v)}{w^*(v) + d^*(v)},$$

where $w^*(v) = \lim_{b\to\infty} \mathrm{E}[W_b(v)]/b$ and $d^*(v) = \lim_{b\to\infty} \mathrm{E}[D_b(v)]/b$.

Proof. According to Lemma 5.3, conditions (a) and (c) imply

$$\lim_{b\to\infty} \frac{W_b(v)}{b} = \lim_{b\to\infty} \frac{\mathrm{E}[W_b(v)]}{b} = w^*(v), \quad a.s.,$$

and similarly, (b) and (d) imply

$$\lim_{b\to\infty} \frac{D_b(v)}{b} = \lim_{b\to\infty} \frac{\mathrm{E}[D_b(v)]}{b} = d^*(v), \quad a.s.$$

According to Lemma 5.1,

$$\bar{p}(v) = \frac{w^*(v)}{w^*(v) + d^*(v)}.$$

The limiting probability is

$$p(v) = \lim_{t\to\infty} p_v(t),$$

which means that there exists a sufficiently large t_0 such that for every $\epsilon > 0$,

$$|p(v) - p_v(t)| < \epsilon, \quad t \geq t_0.$$

Now for $t > t_0$, we have

$$\frac{1}{t}\int_0^t p_v(s)\,ds \leq \frac{1}{t}\int_0^{t_0} p_v(s)\,ds + \frac{1}{t}\int_{t_0}^t p_v(s)\,ds.$$

Hence,

$$\bar{p}(v) = \lim_{t\to\infty} \frac{1}{t}\int_0^t p_v(s)\,ds \leq p(v) + \epsilon.$$

Similarly,

$$\bar{p}(v) \geq p(v) - \epsilon.$$

So we conclude that

$$p(v) = \bar{p}(v) = \frac{w^*(v)}{w^*(v) + d^*(v)},$$

and

$$q(v) = 1 - p(v) = \frac{d^*(v)}{w^*(v) + d^*(v)}.$$

□

In what follows, we present two variants of Theorem 5.4. In the first variant, we use the stronger PA assumption to replace the PQD assumption in Theorem 5.4 (or more precisely, in the underlying Assumption 5.2), which yields that conditions (c) and (d) can be dropped. To see this, we need the following lemma.

Lemma 5.5. [Birkel 89, Theorem 2] *Let $\{S_j(v) : j \geq 1\}$ be a sequence of pairwise PA random variables with finite variance. Assume that*

$$\sum_{j=1}^{\infty} j^{-2} \mathrm{Cov}\left(S_j(v), W_j(v)\right) < \infty.$$

Then as $b \to \infty$, we have $b^{-1}(W_b(v) - \mathrm{E}[W_b(v)]) \to 0$ almost surely.

Assumption 5.6. Assume that $\{S_j(v) : j \geq 1\}$ and $\{C_j(v) : j \geq 1\}$ are sequences of pairwise PA random variables with finite variances, and that

(a) $\sum_{j=1}^{\infty} j^{-2} \mathrm{Cov}\left(S_j(v), W_j(v)\right) < \infty$;

(b) $\sum_{j=1}^{\infty} j^{-2} \mathrm{Cov}\left(C_j(v), D_j(v)\right) < \infty$.

Theorem 5.7. (One variant of Theorem 5.4.) *Under Assumption 5.6 and given a sufficiently large number b of observations on node v's states, the probability $q(v)$, if it exists (i.e., the alternating renewal process is steady), can be expressed as*

$$q(v) = \frac{d^*(v)}{w^*(v) + d^*(v)}.$$

Proof. The proof follows from Lemma 5.5 and the proof of Theorem 5.4. □

In the following we present another useful variant of Theorem 5.4 under another stronger assumption.

Assumption 5.8. Assume that $\{S_j(v) : j \geq 1\}$ and $\{C_j(v) : j \geq 1\}$ are independent sequences with finite variances.

In practice, we can test the independence for sequences $\{S_j : j \geq 1\}$ and $\{C_j : j \geq 1\}$ using the methods described in [Hollander and Douglas 99].

Theorem 5.9. (Another variant of Theorem 5.4.) *Under Assumption 5.8 and given a sufficiently large number b of observations on node v's states, the probability $q(v)$, if it exists (i.e., the alternating renewal process is steady), can be expressed as*

$$q(v) = \frac{d^*(v)}{w^*(v) + d^*(v)}.$$

Proof. The theorem follows from Kolmogorov's strong law of large numbers and the proof of Theorem 5.4. □

The following corollary expresses the connection between Theorem 5.9, which corresponds to the case in which the higher-level alternating renewal process is observed, and Theorem 2.2, which corresponds to the case in which the lower-level attack–defense process is observed.

Corollary 5.10. *Theorem 2.2 is a special case of Theorem 5.9.*

Proof. Under Assumption 2.1 used in Theorem 2.2, we have

$$\begin{aligned} \mathrm{Var}(S(v)) &< 2\,\mathrm{E}\left[\frac{1}{\alpha + K(v)\cdot\gamma}\right]^2 \leq 2\left(\frac{1}{\alpha}\right)^2 < \infty, \\ \mathrm{Var}(C(v)) &= \left(\frac{1}{\beta+\eta}\right)^2 < \infty. \end{aligned}$$

Hence, the conditions in Theorem 5.9 are fulfilled. Therefore, we have

$$w^*(v) = \mathrm{E}[S(v)] = \mathrm{E}\left[\frac{1}{\alpha + \gamma K(v)}\right], \quad d^*(v) = \mathrm{E}[C(v)] = \frac{1}{\beta+\eta}.$$

As a result,

$$q(v) = \mathrm{E}\left[\frac{\beta+\eta}{\alpha + \gamma K(v)}\right],$$

which is the same as in Theorem 2.2. □

5.2. Applying the Analytical Results to Estimating q in Practice

Theorems 5.4, 5.7, and 5.9 require observations only on $\{(S_l(v), C_l(v)); l \geq 1\}$. In order to show how the above analytical results can guide the computation of q in practice, we use Theorem 5.9 as an example. Suppose we select m nodes, denoted by $v_1, \ldots, v_m$, uniformly at random from V to observe their respective $\{(S_l(v), C_l(v)); l \geq 1\}$'s. For each v_i, $1 \leq i \leq m$, we estimate $q(v_i)$ as follows:

1. Observe the node for a sufficiently long time,[1] and get j_i observations $(S_1(v_i), C_1(v_i)), \ldots, (S_{j_i}(v_i), C_{j_i}(v_i))$.
2. Test whether $\{S_l(v_i) : j_i \geq l \geq 1\}$ and $\{C_l(v_i); j_i \geq l \geq 1\}$ are independent sequences (e.g., using methods described in [Hollander and Douglas 99]). If they are independent, the algorithm continues executing the next steps; otherwise, the algorithm aborts.
3. Test whether $\{S_l(v_i) : j \geq l \geq 1\}$ and $\{C_l(v_i); j \geq l \geq 1\}$ have finite variances (e.g., using the methods described in [Resnick 07]). If so, the algorithm continues executing the next step; otherwise, the algorithm aborts.
4. Compute $W_{i,j} = S_1(v_i) + \cdots + S_j(v_i)$ and $D_{i,j} = C_1(v_i) + \cdots + C_j(v_i)$.
5. Compute

$$q(v_i) = \frac{D_{i,j}}{W_{i,j} + D_{i,j}}. \tag{5.1}$$

Having obtained $q(v_1), \ldots, q(v_m)$, we can compute

$$\bar{q} = \frac{\sum_{i=1}^{m} q(v_i)}{m},$$

which, while intuitive, is not unconditionally applicable, as our algorithm demonstrates.

6. Related Work

As extensively discussed in [Li et al. 11], there have been some (promising) attempts at quantitative security analysis of networked systems. In what follows

[1] In practice, we can first observe the time interval $[0, t_1]$ and use (5.1) to compute $q(v_i)$. Then we can observe a longer interval, say $[0, t_2]$, and compute $q(v_i)$ again. Assume that we collect many points $(q(v_i), t_z)$, $z = 1, 2, \ldots$. Then we may plot them to look for a horizontal line. If a horizontal line pattern appears after t_z for some z, we may say that t_z is a sufficiently long time. This method can also be used to determine whether the process is steady in practice.

we discuss three main approaches, which are complementary to the "stochastic process" approach of [Li et al. 11] and the present paper.

An extensively studied approach is the "attack-graph" approach, which was initiated in [Phillips and Swiler 98]. Basically, an attack graph is derived from a networked system and its known vulnerabilities. The nodes in an attack graph are the states of the networked system, and the edges reflect the attack steps (i.e., certain computers having been compromised can cause the compromise of more computers). Along this line, substantial progress has been made in the last decade [Jha and Wing 01, Sheyner et al. 02, Ammann et al. 02, Noel et al. 03, Ingols et al. 06, Ou et al. 06, Wang et al. 07, Sawilla and Ou 08, Wang et al. 08a, Wang et al. 08b, Xie et al. 10, Huang et al. 11]. A particular application of attack graphs is to (optimally) harden specific assets by identifying the relevant attack paths. Our approach is complementary to the attack-graph approach because of the following. First, the attack-graph approach mainly focuses on studying known (but unpatched) vulnerabilities. The only exception we are aware of is the recent extension to considering the number of unknown vulnerabilities that are needed in order to compromise some specific assets [Wang et al. 10]. In contrast, our approach aims to embed unknown vulnerabilities into the model inputs (rather than as an outcome of the model). Second, the attack-graph approach is algorithmic or combinatorial in nature (e.g., computing the number of attack paths or hardening security of a given set of assets with minimal effort). In contrast, our approach aims to model the evolution of node states, which could allow us to derive basic laws or principles for the security of networked systems.

An approach that is closely related to ours is what we call the "dynamical system" approach, in which dynamical system models play an essential role. This approach was rooted in epidemic models [McKendrick 26, Kermack and McKendrick 27], which were first introduced to computer security for studying computer viruses in [Kephart and White 91]. This approach has been coupled with complex network structures since [Wang et al. 03, Ganesh et al. 05, Chakrabarti et al. 08]. The state-of-the-art result in this line of research is [Xu et al. 12], which also presented perhaps the first numerical result for estimating the global state that is comparable to our concept of q. While the results in [Xu et al. 12] can accommodate arbitrary network structures, they are based on an independence assumption that is comparable to that the $X_{2,i}$'s in our model are independent of each other. As shown above, we aim to get rid of this independence and other assumptions as much as we can.

The third approach is what we call the "statistical physics" approach (cf. [Albert and Barabási 02] for a large body of literature and its numerous follow-ons). This approach mainly aims to characterize the robustness of network connectivity (i.e., robustness of giant components in the presence of node and/or edge deletions). While relevant, there is a fundamental difference between network connectivity and security. Specifically, as noted in [Li et al. 11], a malicious attacker may aim to compromise as many computers as possible. However, the ultimate goal of the attacker may be to steal sensitive information rather than to disrupt the connectivity of networks. In other words, an attacker may compromise sensitive information while likely not disrupting the communications of legitimate users.

7. Discussion and Research Directions

In this section we discuss the utility of the stochastic process approach, and suggest future research directions.

7.1. On the Utility of the Stochastic Process Approach

The stochastic process approach, introduced in [Li et al. 11], can be seen as a new way of thinking about quantitative security. The present paper is a first step toward eliminating exponential-distribution assumptions and the independence assumptions in [Li et al. 11]. The approach is centered on stochastic processes that model the interaction between attack and defense. The approach is in its infant stage and is currently based on first-principle modeling (due to the lack of real-life data). Nevertheless, the approach has important real-life implications, as we elaborate below.

First, since stochastic processes capture the evolution of node states, the results are pertinent to the evolution of system states. This will help deepen our understanding of the problem and likely will allow us to draw insights or laws and principles that govern the outcome of the interaction between attack and defense. Note that the characterization is not fundamentally based on knowledge of the parameters, because the parameters exist in any case. In other words, the resulting insights or laws and principles are valid whether we know the specific values of the model parameters or not. Such general (or even universal) laws and principles are of paramount importance.

Second, the analytical results could lead to guidelines (as shown in [Li et al. 11]) for adjusting the defense in a cost-effective, if not optimal, fashion. As mentioned above, the upper bounds can be utilized in practice to capture the worst-case scenario, and can be adopted for decision-making. For example, if we know that q is bounded from above, then some appropriate proactive threshold cryptosystems [Herzberg et al. 97] may be deployed to tolerate the bounded compromise. Moreover, the study can suggest quantitative defense adjustments in order to contain q below a desired threshold (as shown in [Li et al. 11]).

Third, we have offered here statistical methods for obtaining security measures in practice. In Section 4, we presented numerical methods for deriving the global security measure q in various practical settings. In Section 5, we presented statistical methods for obtaining the global security measure q when only high-level information about the attack–defense process can be obtained.

7.2. Extending the Stochastic Process Models

The models in the present paper can be extended in multiple directions. The first direction is to further weaken the assumptions as much as possible. One question is this: how should we accommodate node and edge heterogeneity (other than the degree or topology heterogeneity), meaning that different nodes (edges) exhibit different parameter distribution characteristics? Note that our analytical results on $q(v)$, the probability that node v is compromised in the steady state, already accommodate node and edge heterogeneity. However, our analytical results on the global security measure q, especially its bounds, are based on the fact that the nodes (edges) exhibit the same parameter distribution characteristics.

Another question is this: how can we bridge the gap between what can be observed in practice (i.e., what can be offered by real-life data) and the weakest possible assumptions that we have to make in order to derive analytical results? This includes the design of new statistical methods for testing assumptions that currently cannot be tested.

The second direction is to accommodate multiple vulnerability classes but from a different perspective. Our models are based on aggregating the effects of vulnerabilities at individual computers. Another perspective is to consider individual vulnerability classes that are "compatible" with each other, where "compatibility" is an intuitive concept that needs to be formalized. The intuition is that the exploitation of some vulnerability in one computer can cause the exploitation of another compatible (i.e., not necessarily the same) vulnerability in another computer. This would lead to multiple stochastic attack–defense processes that might be dependent on each other. This quickly becomes very complicated, but certainly worthy of study, because it may lead to deeper insights. Moreover, it is certainly interesting and important to compare the two perspectives.

The third direction is to accommodate the consequence heterogeneity between nodes. This is important because the damage that is caused by the compromise of one node (e.g., a server) may be greater than the damage caused by the compromise of another (e.g., a desktop). Our models can partially accommodate this consequence heterogeneity via the $q(v)$'s, where $q(v)$ is the probability that node v is compromised in the steady state. However, our results on, including the bounds of, the global security measure q do not consider the asset on node v, which may be denoted by asset(v). One simple extension would be to introduce a derivative security measure $q(v) \times \text{asset}(v)$, which can lead to the naturally extended global security measure $\sum_{v \in \mathsf{V}} q(v) \times \text{asset}(v)/|\mathsf{V}|$. Nevertheless, other extensions are possible.

The fourth direction is to characterize the distribution of the random variable that indicates the number of compromised nodes in the steady state. Note that $q \times |\mathsf{V}|$ corresponds to the expected number of compromised nodes in the steady state. Knowledge about the distribution of the random variable will allow us to conduct better decision-making.

7.3. Unifying the Approaches

As mentioned in the introduction, the problem of quantitative security analysis of networked computer systems has been outstanding for decades. In the above section on related work, we highlighted three main approaches. Both the attack-graph approach and the statistical-physics approach have been extensively studied, but both the dynamical system approach and the stochastic process approach are in their infant stages. Nevertheless, it is imperative to unify these approaches into a comprehensive and systematic framework.

8. Conclusion

We extended the quantitative security model and analysis presented in [Li et al. 11] by substantially weakening its assumptions regarding the distributions of, and the dependence between, the relevant random variables. In particular, our extensions lead to practical methods for obtaining the desired security measures for both the case that the lower-level attack–defense process is observed and the case that the higher-level alternating renewal process is observed. We discussed future research directions toward ultimately solving the problem of quantitative security analysis of networked systems.

Acknowledgements. We are very grateful to an anonymous reviewer whose comments helped us improve the paper.

Shouhuai Xu's work was supported in part by grants sponsored by AFOSR, AFOSR MURI, and NSF. The views and conclusions contained in the article are those of the authors and should not be interpreted as in any sense the official policies or endorsements of the United States government or any of its agencies.

References

[Albert and Barabási 02] R. Albert and A. Barabási. "Statistical Mechanics of Complex Networks." *Reviews of Modern Physics* 74 (2002), 47–97.

[Ammann et al. 02] P. Ammann, D. Wijesekera, and S. Kaushik. "Scalable, Graph-Based Network Vulnerability Analysis." In *Proc. ACM Conference Computer and Communications Security (CCS '02)*, pp. 217–224, 2002.

[Barlow and Proschan 81] R. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing*. Silver Spring, MD: To Begin With, 1981.

[Billingsley 95] P. Billingsley. *Probability and Measure*, 3rd edition. New York: Wiley, 1995.

[Birkel 89] T. Birkel. "A Note on the Strong Law of Large Numbers for Positively Dependent Random Variables." *Statistics and Probability Letters* 7 (1989), 17–20.

[Chakrabarti et al. 08] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. "Epidemic Thresholds in Real Networks." *ACM Trans. Inf. Syst. Secur.* 10:4 (2008), 1–26.

[Denuit and Scaillet 04] M. Denuit and Q. Scaillet. "Nonparametric Tests for Positive Quadrant Dependence." *Journal of Financial Econometrics* 2 (2004), 422–450.

[Ganesh et al. 05] A. Ganesh, L. Massoulie, and D. Towsley. "The Effect of Network Topology on the Spread of Epidemics." In *Proc. INFOCOM'05*, pp. 1455–1466, 2005.

[Herzberg et al. 97] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. "Proactive Public Key and Signature Systems." In *ACM Conference on Computer and Communications Security 1997*, pp. 100–110, 1997.

[Hollander and Douglas 99] M. Hollander and A. Douglas. *Nonparametric Statistical Methods*, 2nd edition. Wiley, 1999.

[Huang et al. 11] H. Huang, S. Zhang, X. Ou, A. Prakash, and K. Sakallah. "Distilling Critical Attack Graph Surface Iteratively through Minimum-Cost SAT Solving." In *Proc. ACSAC'11*, pp. 31–40, 2011.

[Ingols et al. 06] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical Attack Graph Generation for Network Defense." In *Proc. of the 22nd Annual Conf. on Computer Security Applications*, pp. 121–130, 2006.

[Jha and Wing 01] S. Jha and J. Wing. "Survivability Analysis of Networked Systems." In *Proc. 23rd Int'l Conf. Software Eng. (ICSE '01)*, pp. 307–317, 2001.

[Johnsonet al. 94] N. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, vol. 1, 2nd edition. New York: Wiley, 1994.

[Kephart and White 91] J. Kephart and S. White. "Directed-Graph Epidemiological Models of Computer Viruses." In *IEEE Symposium on Security and Privacy*, pp. 343–361, 1991.

[Kermack and McKendrick 27] W. Kermack and A. McKendrick. "A Contribution to the Mathematical Theory of Epidemics." *Proc. Roy. Soc. Lond. A* 115 (1927), 700–721.

[Lehmann and Romano 05] E. Lehmann and J. Romano. *Testing Statistical Hypotheses*, 3rd edition. Springer, 2005.

[Li 08] H. Li. “Tail Dependence Comparison of the Survival Marshall–Olkin Copulas.” *Methodology and Computing in Applied Probability* 10 (2008), 39–54.

[Li et al. 11] X. Li, T. Parker, and S. Xu. “A Stochastic Model for Quantitative Security Analysis of Networked Systems.” *IEEE Transactions on Dependable and Secure Computing* 8:1 (2011), 28–43.

[Marlow and Tortorella 95] N. Marlow and M. Tortorella. “Some General Characteristics of Two-State Reliability Models for Maintained Systems.” *Journal of Applied Probability* 32 (1995) 805–820.

[Marshall and Olkin 67] A. Marshall and I. Olkin. “A Multivariate Exponential Distribution.” *Journal of American Statistical Association* 2 (1967), 84–98.

[McKendrick 26] A. McKendrick. “Applications of Mathematics to Medical Problems.” *Proc. Edin. Math. Society* 14 (1926), 98–130.

[Mueller and Stoyan 02] A. Mueller and D. Stoyan. *Comparison Methods for Stochastic Models and Risks.* Chichester: Wiley and Sons, 2002.

[Noel et al. 03] S. Noel, S. Jajodia, B. O’Berry, and M. Jacobs. “Efficient Minimum Cost Network Hardening via Exploit Dependency Graphs.” In *Proc. the 19th Annual Conf. on Computer Security Applications*, pp. 86–95, 2003.

[Ou et al. 06] X. Ou, W. Boyer, and M. McQueen. “A Scalable Approach to Attack Graph Generation.” In *Proc. ACM Conference on Computer and Communications Security 2006*, pp. 336–345, 2006.

[Phillips and Swiler 98] C. Phillips and L. Swiler. “A Graph-Based System for Network-Vulnerability Analysis.” In *Proc. Workshop New Security Paradigms (NSPW ’98)*, pp. 71–79, 1998.

[Provos et al. 07] N. Provos, D. Mcnamee, P. Mavrommatis, K. Wang, and N. Modadugu. “The Ghost in the Browser: Analysis of Web-Based Malware.” In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets (HotBots ’07)*, pp. 4–13, 2007.

[Resnick 07] S. Resnick. *Heavy-Tail Phenomena: Probabilistic and Statistical Modeling.* Springer, 2007.

[Ross 96] S. Ross. *Stochastic Processes.* Wiley and Sons, 1996.

[Sawilla and Ou 08] R. Sawilla and X. Ou. “Identifying Critical Attack Assets in Dependency Attack Graphs.” In *Proc. ESORICS’08*, pp. 18–34, 2008.

[Scaillet 05] Q. Scaillet. “A Kolmogorov–Smirnov Type Test for Positive Quadrant Dependence.” *Canadian Journal of Statistics* 33 (2005), 415–427.

[Sheyner et al. 02] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. “Automated Generation and Analysis of Attack Graphs.” In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 273–284, 2002.

[Stoer and Bulirsch 02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*, 3rd edition. Springer, 2002.

[Wang et al. 03] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos. “Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint.” In *Proc. of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS’03)*, pp. 25–34, 2003.

[Wang et al. 07] L. Wang, A. Singhal, and S. Jajodia. “Measuring the Overall Security of Network Configurations Using Attack Graphs.” In *Prof. DBSec’07*, pp. 98–112, 2007.

[Wang et al. 08a] L. Wang, T. Islam, T. Long, A. Singhal, S. Jajodia. “An Attack Graph-Based Probabilistic Security Metric.” In *Proc. DBSec’08*, pp. 283–296, 2008.

[Wang et al. 08b] L. Wang, C. Yao, A. Singhal and S. Jajodia. “Implementing Interactive Analysis of Attack Graphs Using Relational Databases.” *Journal of Computer Security* 16:4 (2008), 419–437.

[Wang et al. 10] L. Wang, S. Jajodia, A. Singhal, and S. Noel. “k-Zero Day Safety: Measuring the Security Risk of Networks against Unknown Attacks.” In *Proc. ESORICS’10*, pp. 573–587, 2010.

[Xie et al. 10] P. Xie, J. Li, X. Ou, P. Liu, and R. Levy. “Using Bayesian Networks for Cyber Security Analysis.” In *Proc. DSN’10.*, pp. 211–220, 2010.

[Xu et al. 12] S. Xu, W. Lu, and L. Xu. “Push- and Pull-Based Epidemic Spreading in Networks: Thresholds and Deeper Insights.” To appear in *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2012.

Maochao Xu, Department of Mathematics, Illinois State University, Normal, IL 61790-4520, USA (mxu2@ilstu.edu)

Shouhuai Xu, Department of Computer Science, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA (shxu@cs.utsa.edu)

L-hop percolation on networks with arbitrary degree distributions and its applications

Yilun Shang,[1] Weiliang Luo,[2] and Shouhuai Xu[2]

[1]*Institute for Cyber Security, University of Texas at San Antonio, Texas 78249, USA*

[2]*Department of Computer Science, University of Texas at San Antonio, Texas 78249, USA*

(Received 22 March 2011; revised manuscript received 4 July 2011; published xxxxx)

Site percolation has been used to help understand analytically the robustness of complex networks in the presence of random node deletion (or failure). In this paper we move a further step beyond random node deletion by considering that a node can be deleted because it is chosen or because it is within some L-hop distance of a chosen node. Using the generating functions approach, we present analytic results on the percolation threshold as well as the mean size, and size distribution, of nongiant components of complex networks under such operations. The introduction of parameter L is both conceptually interesting because it accommodates a sort of *nonindependent* node deletion, which is often difficult to tackle analytically, and practically interesting because it offers useful insights for cybersecurity (such as botnet defense).

DOI: 10.1103/PhysRevE.00.001100 PACS number(s): 02.50.−r, 64.60.−i, 89.75.Hc

I. INTRODUCTION

Understanding the robustness (or connectivity) of complex networks in the presence of node deletion (or failure) is an important problem that has attracted much attention. Existing analytic studies mainly focused on the case of randomly deleting nodes in random graphs with a given arbitrary degree distribution (the so-called configuration model [1–5]). By treating node deletion as site percolation on large networks (i.e., in the limit of network size), Newman *et al.* [3] introduced a novel generating functions approach to tackling the problem. This approach has been used in various settings (e.g., Refs. [6–8]).

In this paper, we move a further step beyond random node deletion by considering the following node deletion operation: a node is deleted either because it is chosen or because it is within a chosen node's L-hop distance in terms of shortest path between them in unweighted graphs, where $L \geqslant 0$. Figure 1 illustrates an example scenario with $L = 1$, where, if v is chosen to be deleted, then both v and its one-hop (i.e., direct) neighbors u_1,u_2,u_3,u_4 are deleted. We call the new model "L-hop percolation" because the random node deletion extensively studied in the literature [3–8] can be seen as the special case with $L = 0$. The introduction of the new parameter L is of both conceptual and practical value, as we discuss below.

From a conceptual perspective, the introduction of L allows us to model and characterize a class of *nonindependent* node deletion operations. Previous analytical studies (including Refs. [3–8]) in this context focused on *independent* node deletions. Nonindependent node deletions are often challenging to treat analytically, but we manage to present some analytical results in the specific nonindependence setting of this paper.

From a practical perspective, the nonindependent node deletion has real-life applications. On the one hand, an adversary could certainly destroy both randomly chosen nodes and their neighbors within L-hop distance. On the other hand, analyzing such nonindependent node deletions can lead to useful insights for defending cyberspace. To see this, consider an example of botnet defense. A botnet is a network of computers (called bots or zombies) that are compromised by a computer malware/virus and under the control of a bot-master (the human attacker). Peer-to-peer (P2P) botnets [9] are arguably the most powerful cyberattacks that are very difficult to defend against. This is because a bot only "knows" (i.e., can directly communicate with) a few others, which suffices their needs of forwarding the bot-master's command-and-control messages. In real life, it is very difficult to obtain the precise structures of P2P botnets because of various technical issues and ethics/privacy concerns [10,11]. This means that the defender cannot expect to identify all the bots belonging to a botnet and then eliminate them as a whole. Instead, it is more realistic that the defender, first, detects some bots and then traces the bots that communicated with the detected ones (we call this method "detect and then trace"). However, the defender's capability in tracing bots are limited by various reasons. For example, botmasters have abused cryptographic techniques for making P2P botnets—such as Conficker, Nugache, and Storm and its successor, Waledac—more stealthy [9,12–15]. Moreover, the Waledac botnets actually do not spread (propagate) using a scanning technique that is relatively easy to defend against but rather spread (propagate) via social-engineering methods that are much more difficult to deal with [16]. Because P2P botnets are naturally complex networks, the defender's limited tracing capability in combating P2P botnets naturally motivates the concept of L-hop percolation. Note that $L = 0$ corresponds to the fact that the defender can only directly detect (and thus delete) some bots, and $L \geqslant 0$ corresponds to the fact that the defender not only can directly detect (and thus delete) some bots but also can trace to (and thus delete) the bots within their L-hop neighborhood.

The present paper aims to understand and characterize the robustness of complex networks with $L > 0$ when compared to the case with $L = 0$. Specifically, we consider L-hop site percolation on random graphs with arbitrary node degree distributions (in the configuration model). Using the generating functions approach [3,6], we present analytical results on the percolation threshold, and mean size as well as size distribution of the nongiant components. The analytical results are confirmed by our simulation study on power-law random graphs and Erdos-Renyi random graphs.

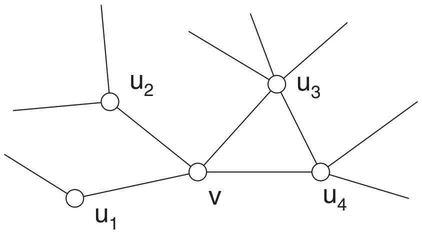


FIG. 1. An example scenario: when $L = 0$, a randomly chosen node v is deleted (this is the case extensively studied in the literature); when $L = 1$, a randomly chosen node v and its one-hop neighbors u_1,u_2,u_3,u_4 are deleted. As defined in the text, $\mathcal{N}_{\leqslant 0}(v) = \{v\}$ and $\mathcal{N}_{\leqslant 1}(v) = \{v,u_1,u_2,u_3,u_4\}$.

II. SYSTEM MODEL AND ANALYTIC RESULTS

Let $\mathsf{G} = (\mathsf{V},\mathsf{E})$ be a complex network (or graph), where $v \in \mathsf{V}$ is the set of nodes (vertices) and E is the set of edges (we focus on undirected unweighted graphs). Let $\mathsf{distance}(u,v)$ be the length of the shortest path between $u \in \mathsf{V}$ and $v \in \mathsf{V}$ in G, where $\mathsf{distance}(v,v) = 0$. Define the ℓ-hop neighborhood of node v as $\mathcal{N}_{\leqslant \ell}(v) = \{u : u \in \mathsf{V}, \mathsf{distance}(u,v) \leqslant \ell\}$, namely the set of nodes that are no more than ℓ-hop away from v. The following parameters are used in our model:

(1) L: This is upper bound on the number of hops the deletion operation can reach out to when starting at a randomly chosen node. Note that L indicates, in a sense, the degree of nonindependence of the nodes that are deleted (intuitively, the larger the L, the stronger the nonindependence) and can represent (for instance) the defender's capability in tracing bots in the above example of botnet defense.

(2) $(r_0,r_1,\ldots,r_L)$ with $\sum_{\ell=0}^{L} r_\ell = 1$: For a node v and its L-hop neighborhood $\mathcal{N}_{\leqslant L}(v)$, the simplest scenarios is to delete all the nodes belonging to $\mathcal{N}_{\leqslant \ell}(v)$, which is a special case we mentioned above. To make our model more generally applicable while reflecting the fact mentioned above—the farer from v the neighbors, the less likely the defender can trace to them—we need a representation of detection probability that decreases as the distance from a chosen node grows. For this, we use vector $(r_0,r_1,\ldots,r_L)$ to represent a class of deletion "strategies" such that a node u, which is $\mathsf{distance}(u,v)$-hop away from a chosen node v, will be deleted with probability $\sum_{\ell=\mathsf{distance}(u,v)}^{L} r_\ell$. Because the chosen node v will be deleted with probability $\sum_{\ell=0}^{L} r_\ell$, we naturally require $\sum_{\ell=0}^{L} r_\ell = 1$, meaning that v will always be deleted. This explains why the extensively studied case with $L = 0$ is a special case of the model with $L \geqslant 0$ in this paper. In order to see that the representation corresponds to a class of deletion operations, we note that the scenario of $L = 0$ shown in Fig. 1 corresponds to strategy $(r_0) = (1)$, and the scenario of $L = 1$ corresponds to strategy $(r_0,r_1) = (0,1)$. Continuing on the example of $L = 1$ but with a different strategy $(0 < r_0 < 1, 0 < r_1 < 1)$ where $r_0 + r_1 = 1$, we observe that node v will be deleted with probability $r_0 + r_1 = 1$ but every node belonging to $\mathcal{N}_{\leqslant 1}(v) - \mathcal{N}_{\leqslant 0}(v) = \{u_1,u_2,u_3,u_4\}$ will be deleted with probability r_1.

(3) p_k: This is the probability that a randomly chosen node has degree k.

(4) q, q_k, q_c: q is the probability an appropriately chosen node is *occupied* (i.e., not deleted), and $1 - q$ is the probability that node is *unoccupied* (i.e., deleted). In general $0 \leqslant q \leqslant 1$ because, for example, the curing of a bot may not be perfect [17]. Correspondingly, q_k is the probability that a node of degree k is occupied, and q_c is the percolation threshold we want to derive, above which a giant component emerges.

Suppose $|x| \leqslant 1$. Following the generating functions approach [3], $G_0(x) = \sum_{k=0}^{\infty} p_k x^k$ and $G_1(x) = G_0'(x)/G_0'(1)$ are the generating functions of nodes' degree distribution and nodes' excess-degree distribution, respectively. For any node $v \in \mathsf{V}$, the generating function for the size distribution of $\mathcal{N}_{\leqslant \ell}(v) - \mathcal{N}_{\leqslant \ell-1}(v)$ is $G^{(\ell)}(x) := G_0[G_1(\cdots G_1(x)\cdots)]$, with $\ell - 1$ iterations of the function G_1 acting on itself. For a given L, for each $0 \leqslant \ell \leqslant L$ we can define a "base" strategy $\mathsf{baseStrategy}_\ell = (\mathsf{r}_0,\ldots,\mathsf{r}_L)$, where $\mathsf{r}_j = 1$ if $j = \ell$ and $\mathsf{r}_j = 0$ otherwise. (This is for deriving some intermediate formulas that will be used to derive the final result.) Let $q_k^{(\ell)}$ be the probability that a node of degree k is occupied (i.e., not deleted) with respect to $\mathsf{baseStrategy}_\ell$. Note that the component structure in graph G is essentially treelike because the probability that a component contains a closed loop goes as N^{-1} and is therefore negligible in the limit of large N. Then, for a node v of degree k, we have $q_k^{(0)} = q$, $q_k^{(1)} = q^{k+1}$, and for $\ell \geqslant 2$,

$$q_k^{(\ell)} = \sum_{s_1=0}^{\infty}\cdots\sum_{s_{\ell-1}=0}^{\infty}\left(q^{k+1+\sum_{j=1}^{\ell-1}s_j}\prod_{j=1}^{\ell-1}\left.\frac{d^{s_j}G^{(j+1)}(x)}{s_j!\,dx^{s_j}}\right|_{x=0}\right), \quad (1)$$

where s_j is the number of v's distance-$(j+1)$ neighbors [i.e., s_j is the size of the set $\{u : \mathsf{distance}(u,v) = j + 1\}$], $\frac{d^{s_j}G^{(j+1)}(x)}{s_j!dx^{s_j}}|_{x=0}$ is the probability that v has exactly s_j distance-$(j+1)$ neighbors (based on the property of derivatives of generating functions for degree distributions [3,18]), $\prod_{j=1}^{\ell-1}\frac{d^{s_j}G^{(j+1)}(x)}{s_j!dx^{s_j}}|_{x=0}$ is the probability that v has respectively s_j distance-$(j+1)$ neighbors for $j = 1,\ldots,\ell-1$, and $q^{k+1+\sum_{j=1}^{\ell-1}s_j}\prod_{j=1}^{\ell-1}\frac{d^{s_j}G^{(j+1)}(x)}{s_j!\,dx^{s_j}}|_{x=0}$ is the probability that v is occupied (i.e., not deleted) with respect to $\mathsf{baseStrategy}_\ell$ for a specific ℓ.

Given $q_k^{(\ell)}$ for $\ell = 0,1,\ldots$ and strategy $(r_1,\ldots,r_L)$, we obtain

$$q_k = \sum_{\ell=0}^{L} q_k^{(\ell)} r_\ell. \quad (2)$$

The probability that any node of degree k is occupied is given by $p_k q_k$, and

$$F_0(x) = \sum_{k=0}^{\infty} p_k q_k x^k \quad (3)$$

is the probability generating function for this distribution [18]. Since $F_0(1)$ is the probability that a randomly chosen node is occupied, it can be viewed as the portion of occupied nodes.

As in Refs. [3,6], if we start at a randomly chosen node and follow each of its edges to reach its neighbors, the distribution of the other edges of each node arrived and occupied is generated by

$$F_1(x) = \frac{\sum_{k=0}^{\infty} k p_k q_k x^{k-1}}{\sum_{k=0}^{\infty} k p_k} = \frac{F_0'(x)}{z}, \quad (4)$$

where $z = \langle k \rangle = G_0'(1)$ is the average node degree.

Let $H_1(x)$ be the generating function for the distribution of the sizes of percolation components with occupied nodes that are reached by choosing a random edge and following it to one of its ends. With the same sort of reasoning as in Ref. [7], we derive that $H_1(x)$ satisfies the self-consistency condition

$$H_1(x) = 1 - F_1(1) + xF_1[H_1(x)]. \tag{5}$$

Similarly, the distribution for the size of the component to which a randomly chosen node belongs is generated by

$$H_0(x) = 1 - F_0(1) + xF_0[H_1(x)], \tag{6}$$

which, in conjunction with Eqs. (1)–(5), can be used to determine some quantities of interest such as the L-hop percolation threshold, the mean size, and size distribution of nongiant components. In what follows we present the details.

A. L-hop percolation threshold

From Eqs. (5) and (6), we obtain that in the absence of giant components, the mean size of components to which a randomly chosen node belongs is given by

$$\begin{aligned}\langle s\rangle &= H_0'(1) = F_0(1) + F_0'(1)H_1'(1)\\ &= F_0(1) + \frac{F_0'(1)^2}{z - F_0''(1)}.\end{aligned} \tag{7}$$

The above expression diverges at $z = F_0''(1)$, which corresponds to the percolation threshold q_c (i.e., the critical occupation probability) where a giant component emerges. Hence, q_c is determined by

$$\sum_{k=0}^{\infty} kp_k = \sum_{k=0}^{\infty} k(k-1)p_kq_k, \tag{8}$$

where q_k is given by Eq. (1). In what follows we instantiate the above general q_c in the special cases of regular graphs and of the classical Erdos-Renyi random graphs.

In the case of d-regular graphs G on n nodes, where d is a positive integer, we have the degree distribution $p_d = 1$ and $p_k = 0$ for $k \neq d$. From Eqs. (1) and (2) we get

$$q_d = \sum_{\ell=0}^{L} q_d^{(\ell)} r_\ell, \tag{9}$$

where for $\ell = 0, \ldots, L$,

$$q_d^{(\ell)} = \sum_{s_1=0}^{\infty} \cdots \sum_{s_{\ell-1}=0}^{\infty} \left(q^{d+1+\sum_{j=1}^{\ell-1} s_j} \prod_{j=1}^{\ell-1} \frac{d^{s_j} G^{(j+1)}(x)}{s_j!\, dx^{s_j}}\bigg|_{x=0} \right). \tag{10}$$

By utilizing Eq. (8), we see that the percolation threshold q_c can be calculated from

$$1 = (d-1)q_d. \tag{11}$$

In the case of the classical Erdos-Renyi random graphs $G(n,\lambda/n)$ for some $\lambda > 0$ [19], the average degree for each vertex is λ and the degree distribution is given by $p_k = e^{-\lambda}\lambda^k/k!$ for $k \geqslant 0$. From Eq. (8), we see that the percolation threshold q_c is determined by

$$\sum_{k=2}^{\infty} \frac{\lambda^k}{(k-2)!}\left(\frac{1}{k-1} - q_k\right) = 0. \tag{12}$$

where q_k is given by (2) and (1).

B. Mean size of nongiant components

Because $H_0(x)$ generates the size distribution of nongiant components, $H_0(1)$ takes the value of $1 - S$, where S is the fraction of nodes belonging to some giant component (equivalently, S is the fraction of giant components). Therefore,

$$S = 1 - H_0(1) = F_0(1) - F_0(u), \tag{13}$$

where $u = H_1(1)$ is the smallest non-negative solution of

$$u = 1 - F_1(1) + F_1(u). \tag{14}$$

In general, the mean size of the component, excluding the infinite giant component, to which a randomly chosen vertex belongs can be expressed as

$$\begin{aligned}\langle s\rangle &= \frac{H_0'(1)}{H_0(1)} = \frac{F_0[H_1(1)] + F_0'[H_1(1)]\left\{\frac{F_1[H_1(1)]}{1-F_1'[H_1(1)]}\right\}}{H_0(1)}\\ &= \frac{F_0(u)[z - F_0''(1)] + F_0'(u)^2}{(1-S)[z - F_0''(1)]},\end{aligned} \tag{15}$$

which is equivalent to Eq. (7) when there is no giant component (i.e., $S = 0$, $u = 1$).

C. Size distribution of nongiant components

Let π_s be the probability of a randomly chosen node belonging to a component of size s. We can then rewrite the generating function $H_0(x)$ as

$$H_0(x) = \pi_0 + \sum_{s=1}^{\infty} \pi_s x^s. \tag{16}$$

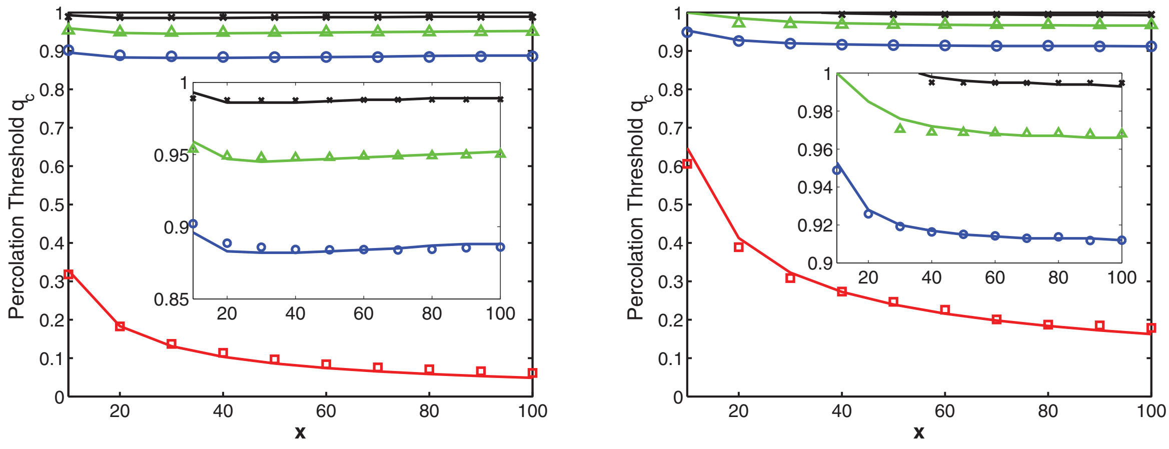
It is straightforward to see that $\pi_0 = 1 - F_0(1)$ and $\pi_1 = H_0(1) - \pi_0 - \sum_{s=2}^{\infty} \pi_s$, where $H_0(1)$ can be determined by Eqs. (13) and (14). For $s \geqslant 2$, we have from Eqs. (16) and (6)

$$\begin{aligned}\pi_s &= \frac{1}{(s-1)!}\left\{\frac{d^{s-1}}{dx^{s-1}}\left[\frac{H_0(x) - \pi_0}{x}\right]\right\}\bigg|_{x=0}\\ &= \frac{1}{(s-1)!}\frac{d^{s-1}}{dx^{s-1}}\{F_0[H_1(x)]\}|_{x=0}\\ &= \frac{1}{(s-1)!}\frac{d^{s-2}}{dx^{s-2}}[F_0'(H_1(x))H_1'(x)]|_{x=0}.\end{aligned} \tag{17}$$

By using the Cauchy formula for the higher-order derivative of a function [20],

$$\frac{d^nf}{dz^n}\bigg|_{z=z_0} = \frac{n!}{2\pi i}\oint \frac{f(z)}{(z-z_0)^{n+1}}dz, \tag{18}$$

where the integral is around a contour that encloses z_0 in the complex plane but encloses no poles in $f(z)$. Equation (17)



(a) $t = 2$ (b) $t = 2.5$

FIG. 2. (Color online) Percolation threshold q_c in the case of power-law graphs. Simulation results are plotted with respect to $L = 0$ with strategy $(r_0) = (1)$ (squares), $L = 1$ with strategy $(r_0,r_1) = (0,1)$ (circles), $L = 2$ with strategy $(r_0,r_1,r_2) = (0,0,1)$ (triangles), and $L = 3$ with strategy $(r_0,r_1,r_2,r_3) = (0,0,0,1)$ (crosses). The insets show a magnified view of the curves for $L = 1$, $L = 2$, and $L = 3$. In the simulation we set the threshold for giant component as 1000 nodes. The plots of simulation results correspond to the average of 100 independent simulation runs. Exact solutions (solid curves) are obtained from Eq. (8).

then becomes

$$\pi_s = \frac{1}{2\pi i(s-1)}\oint \frac{F_0'[H_1(x)]}{x^{s-1}}\frac{dH_1(x)}{dx}dx \tag{19}$$

$$= \frac{z}{2\pi i(s-1)}\oint \frac{F_1(H_1)}{x^{s-1}}dH_1, \tag{20}$$

where the integral in Eq. (19) is around an infinitesimal contour around the origin in the complex plane, and the integral in Eq. (20) is an infinitesimal loop around $1 - F_1(1)$ since $H_1(x) \to 1 - F_1(1)$ as $x \to 0$.

By treating x as a function of H_1, we plug Eq. (5) into Eq. (20) to obtain

$$\pi_s = \frac{z}{2\pi i(s-1)}\oint \frac{[F_1(H_1)]^s}{[H_1 - 1 + F_1(1)]^{s-1}}dH_1. \tag{21}$$

By applying the Cauchy formula to Eq. (21), we obtain the desired result

$$\pi_s = \frac{z}{(s-1)!}\left\{\frac{d^{s-2}}{dx^{s-2}}[F_1(x)^s]\right\}\Bigg|_{x=1-F_1(1)}. \tag{22}$$

III. SIMULATION RESULTS

A. Simulation-based confirmation of the percolation thresholds

In order to verify the theoretical results, we first conducted simulations on random graphs at the order 1 000 000 nodes with node degrees distributed according to the truncated power law $p_0 = 0$ and

$$p_k = Ck^{-t}e^{-k/x}, \qquad k \geqslant 1, \tag{23}$$

where C, t, and x are constants. The reason for choosing this distribution is twofold. On one hand, this distribution is seen in various real-life networks including the Internet [21], collaboration networks of movie actors and scientists [22,23]. On the other hand, the exponential cutoff makes the distribution normalizable for all t so the generating functions and their derivatives are finite. In our simulation study we set $0 \leqslant L \leqslant 3$ and, for simplicity, strategy $(r_0,\ldots,r_L) = (0,\ldots,0,1)$. We consider two scenarios of $t = 2$ and $t = 2.5$.

Figure 2 plots the percolation threshold q_c obtained from our simulations, along with the exact solutions obtained from Eq. (8). We make the following observations. First, the simulation results agree with their analytical counterparts. Second, there is a huge gap between the percolation thresholds

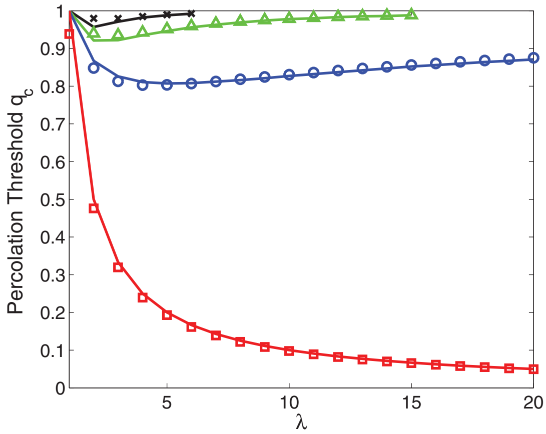
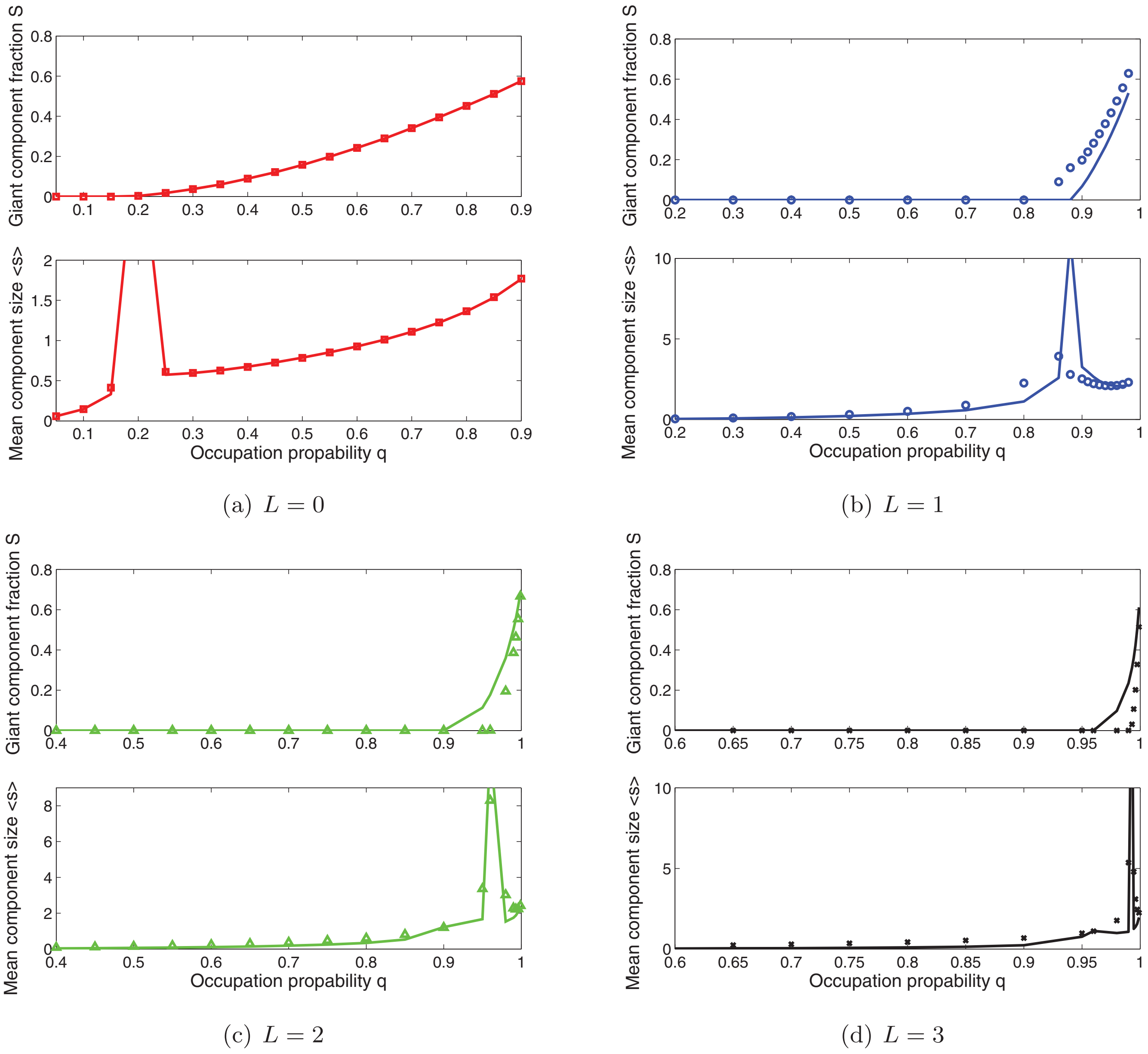


FIG. 3. (Color online) Percolation threshold in the case of Erdos-Renyi graphs. Simulation results are plotted with respect to $L = 0$ with strategy $(r_0) = (1)$ (squares), $L = 1$ with strategy $(r_0,r_1) = (0,1)$ (circles), $L = 2$ with strategy $(r_0,r_1,r_2) = (0,0,1)$ (triangles), and $L = 3$ with strategy $(r_0,r_1,r_2,r_3) = (0,0,0,1)$ (crosses). In the simulation we set the threshold for giant component as 1000 nodes. The plots of simulation results correspond to the average of 100 independent simulation runs. Exact solutions (solid curves) are obtained from Eq. (12).

with respect to $L = 0$ and $L = 1$. For example, consider the case of $t = 2$ with $x = 50$. When $L = 0$, giant component disappears only after randomly deleting 90% nodes; whereas, when $L = 1$, giant component disappears after deleting 12% nodes, including both the randomly chosen nodes and their direct neighbors. This means, in the botnet defense example, that "detect and then trace" could be much more effective.

Third, there is no big difference between the thresholds corresponding to $L = 1,2,3$, respectively. This phenomenon can be interpreted as follows: In networks with a power-law degree distribution, most nodes have small degrees and only a small number of nodes (called hubs) have large degrees. Therefore, a randomly chosen node probably has a neighbor that is a hub, meaning that when $L = 1$ with strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$, a hub could be deleted even after choosing a small number of nodes. This explains the abrupt change in the percolation threshold in the case of $L = 1$ with strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$ when compared with the percolation threshold in the case of $L = 0$ with $\mathrm{r}_0 = 1$. The fact that $L = 1$ already allows the deletion of some hub nodes further explains why increasing L further may not significantly increase the percolation threshold. This means that tracing one-hop neighbors is almost as powerful as tracing multihop neighbors. We mention that $L = 1$ with



(a) $L = 0$

(b) $L = 1$

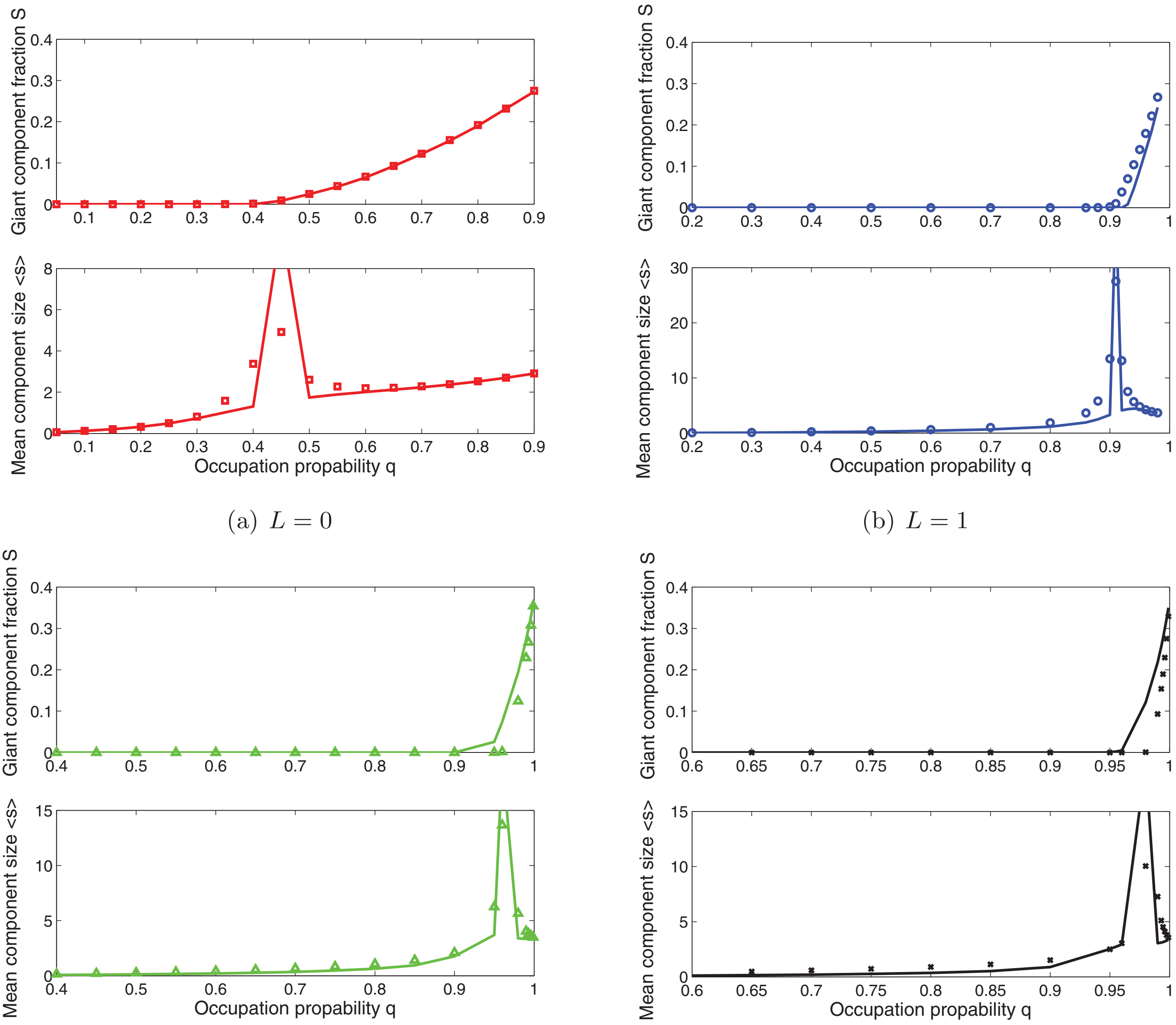
(c) $L = 2$

(d) $L = 3$

FIG. 4. (Color online) Fraction of giant components S and mean size of nongiant components $\langle s \rangle$ in power-law random graphs with $t = 2$ and $x = 20$. Simulation results are plotted with respect to (a) $L = 0$ with strategy $(\mathrm{r}_0) = (1)$ (squares), (b) $L = 1$ with strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$ (circles), (c) $L = 2$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2) = (0,0,1)$ (triangles), and (d) $L = 3$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2,\mathrm{r}_3) = (0,0,0,1)$ (crosses). In the simulation, we set the threshold for giant component as 1000 nodes. The plots of simulation results correspond to the average of 100 independent simulation runs. Exact solutions (solid curves) are obtained from Eqs. (13), (14), and (15).

strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$ is somewhat related to the method known as "acquaintance immunization" [24], which consists of choosing a random node and deleting a random one-hop neighbor of the chosen node (the chosen node itself is not deleted). It was shown in Ref. [24] that the effectiveness of this method (especially on power-law networks) comes from the fact that with high probability the randomly chosen neighbor of a randomly chosen node is a hub, and therefore the hubs are knocked out after a small number of such operations. In this paper, the same effect causes the dramatic increase in q_c when changing $L = 0$ to $L = 1$.

We then conducted simulations on the classic Erdos-Renyi graphs $G(n,p)$ with $n = 1\,000\,000$ nodes and link probability $p = \lambda/n$. Figure 3 plots the percolation threshold q_c obtained from our simulations, along with the exact solutions obtained from Eq. (12). Similar qualitative behavior can be observed as in the case of power-law graphs, with two noticeable differences: the difference between the thresholds corresponding to $L = 0$ and $L = 1,2,3$ trails off, while the differences between $L = 1,2,3$ become more apparent. These phenomena can be explained by that Erdos-Renyi random graphs have more homogeneous degree distributions than power-law graphs.



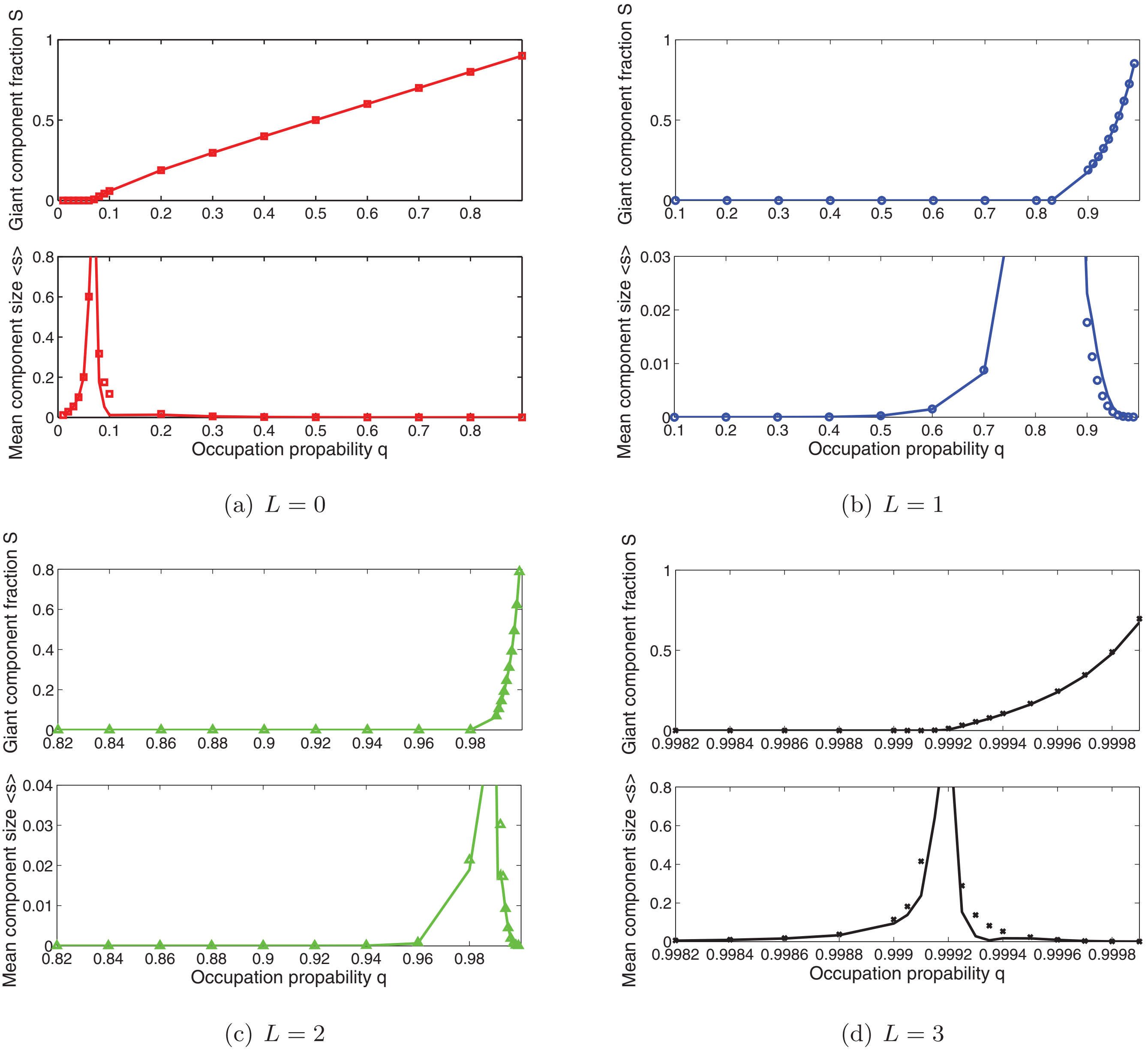
(a) $L = 0$

(b) $L = 1$

(c) $L = 2$

(d) $L = 3$

FIG. 5. (Color online) Fraction of giant components S and mean size of nongiant components $\langle s\rangle$ in power-law random graphs with $t = 2.5$ and $x = 20$. Simulation results are plotted with respect to (a) $L = 0$ with strategy $(\mathrm{r}_0) = (1)$ (squares), (b) $L = 1$ with strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$ (circles), (c) $L = 2$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2) = (0,0,1)$ (triangles), and (d) $L = 3$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2,\mathrm{r}_3) = (0,0,0,1)$ (crosses). In the simulation, we set the threshold for giant component as 1000 nodes. The plots of simulation results correspond to the average of 100 independent simulation runs. Exact solutions (solid curves) are obtained from Eqs. (13), (14), and (15).



(a) $L = 0$

(b) $L = 1$

(c) $L = 2$

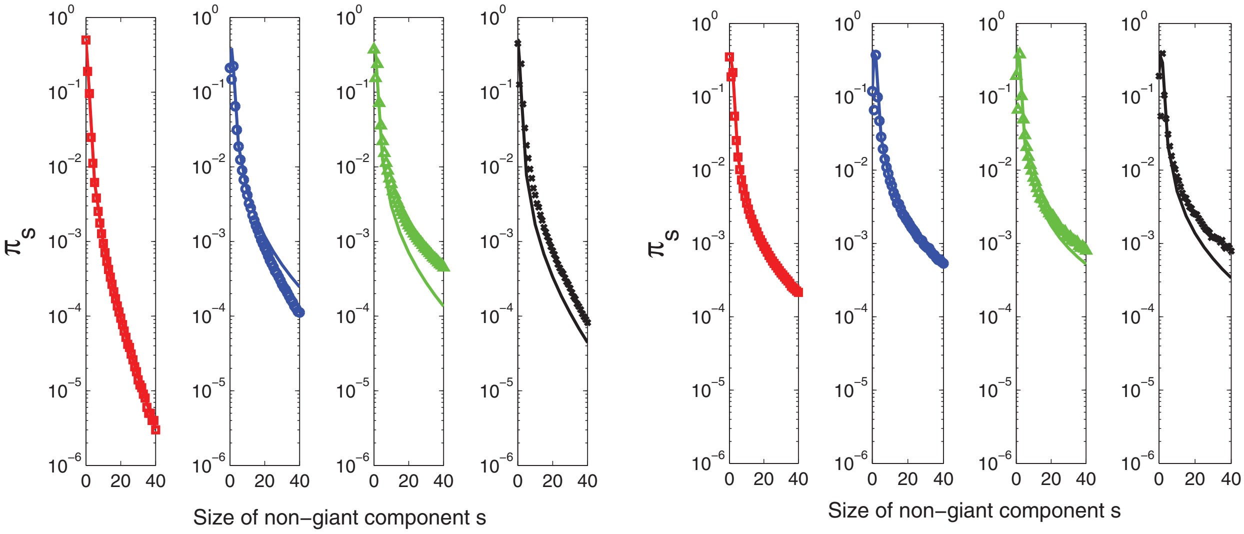
(d) $L = 3$

FIG. 6. (Color online) Fraction of giant components S and mean size of nongiant components $\langle s \rangle$ in Erdos-Renyi random graphs with $\lambda = 15$. Simulation results are plotted with respect to (a) $L = 0$ with strategy $(\mathrm{r}_0) = (1)$ (squares), (b) $L = 1$ with strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$ (circles), (c) $L = 2$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2) = (0,0,1)$ (triangles), and (d) $L = 3$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2,\mathrm{r}_3) = (0,0,0,1)$ (crosses). In the simulation we set the threshold for giant component as 1000 nodes. The plots of simulation results correspond to the average of 100 independent simulation runs. Exact solutions (solid curves) are obtained from Eqs. (13), (14), and (15).

B. Simulation-based confirmation of the fraction S of giant components and the mean size $\langle s \rangle$ of the nongiant components

For this purpose, we conducted simulations on the same power-law random graphs with distribution (23) and the same Erdos-Renyi random graphs $G(n,\lambda/n)$ with $n = 1\,000\,000$. Figure 4 and Fig. 5 plot the cases of power-law graphs with $t = 2, x = 20$ and $t = 2.5, x = 20$, respectively. Figure 6 plots the case of Erdos-Renyi random graphs with $\lambda = 15$. From them we draw the following observations.

First, we observe that when the occupation probability $q < q_c$, the fraction of giant components is $S = 0$. Then, S increases with q when $q \geqslant q_c$ for all four strategies $(r_0, \ldots, r_L) = (0, \ldots, 0, 1)$ with respect to $L = 0, 1, 2, 3$. More interestingly, from another perspective, we observe that as the number of chosen nodes increases, the fraction of giant components decreases abruptly in the cases of $L > 1$ but decreases smoothly in the case of $L = 0$. This illustrates the power of deletion of even one-hop neighbors. Moreover, the giant component disappears faster in the case of power-law graphs than in the case of Erdos-Renyi random graphs. For example, in the case of $L = 1$, the giant component disappears after deleting about 10% (20%) of nodes and their one-hop neighbors in the case power-law graphs with $t = 2.5$ as shown in Fig. 5(b) [in the case of Erdos-Renyi random graphs as shown in Fig. 6(b)]. This can be explained, as mentioned above,



(a) Power-law graphs with $t = 2, x = 20$ (b) Power-law graphs with $t = 2.5, x = 20$

FIG. 7. (Color online) Size distribution for nongiant components π_s in power-law random graphs. Simulation results are plotted with respect to, from left to right, $L = 0$ with strategy $(\mathrm{r}_0) = (1)$ (squares), $L = 1$ with strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$ (circles), $L = 2$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2) = (0,0,1)$ (triangles), and $L = 3$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2,\mathrm{r}_3) = (0,0,0,1)$ (crosses). In the simulation we set in (a) the occupation probabilities $q = 0.5,0.92,0.95,0.98$ to keep the fraction of giant components $S \approx 0.11$ in all four cases; we set in (b) $q = 0.65,0.95,0.96,0.98$ to keep the fraction of giant components $S \approx 0.09$ in all four cases. The simulation results correspond to the average of 100 independent simulation runs. Exact solutions (solid curves) are obtained from Eq. (22).

by the fact that in the case of power-law graphs, it is relatively easier to hit the hubs because they are the neighbors of many nodes.

Second, when q approaches the critical point q_c, the mean size of nongiant components $\langle s\rangle$ increases abruptly. This agrees with our theory. In fact, when $q = q_c$ we obtain $z = F_0''(1)$ in Eq. (21). By using Eq. (15), we obtain $\lim_{q\to q_c}\langle s\rangle = \infty$. The percolation thresholds q_c shown in Fig. 4, Fig. 5, and Fig. 6 agree with those in Fig. 2(a), Fig. 2(b), and Fig. 3. By comparing Fig. 4 and Fig. 5 with Fig. 6, we make two observations. (i) In power-law graphs, typical nongiant components have multiple nodes. In Erdos-Renyi random graphs, however, most nongiant components are isolated nodes because $\langle s\rangle$ is always less than 1 (when unoccupied nodes are treated as components of size zero). (ii) The analytical and numerical results match better in the case of Erdos-Renyi random graphs than in the case of power-law graphs, due to the inhomogeneity of power-law degree distribution.

C. Simulation-based confirmation of the size distribution π_s of nongiant components

To illustrate this, we conducted simulations on the same power-law random graphs with distribution (23) and the same Erdos-Renyi random graphs $G(n,\lambda/n)$ with $n = 1\,000\,000$. We plot the log-linear diagrams in Fig. 7 for power-law graphs and in Fig. 8 for Erdos-Renyi random graphs.

In Fig. 7(a) and Fig. 7(b), we keep the respective fractions of giant components at $S \approx 0.11$ and $S \approx 0.09$ in all strategies $L = 0,1,2,3$ (this can be achieved by choosing different occupation probabilities q). In Fig. 8, we keep the fraction of giant components at $S \approx 0.07$ in all strategies $L = 0,1,2,3$ (also by choosing different occupation probabilities q).

By comparing Fig. 7 and Fig. 8, we observe the following discrepancy between power-law graphs and Erdos-Renyi graphs. For power-law graphs with the same fraction of

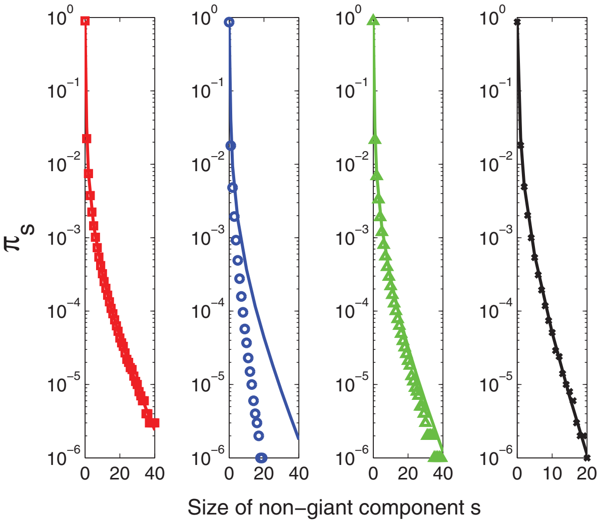


FIG. 8. (Color online) Size distribution for nongiant components π_s in Erdos-Renyi random graphs with $\lambda = 15$. Simulation results are plotted with respect to, from left to right, $L = 0$ with strategy $\mathrm{r}_0 = 1$ (squares), $L = 1$ with strategy $(\mathrm{r}_0,\mathrm{r}_1) = (0,1)$ (circles), $L = 2$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2) = (0,0,1)$ (triangles), and $L = 3$ with strategy $(\mathrm{r}_0,\mathrm{r}_1,\mathrm{r}_2,\mathrm{r}_3) = (0,0,0,1)$ (crosses). In the simulation we set the occupation probabilities $q = 0.1,0.87,0.99,0.994$ to keep the fraction of giant components $S \approx 0.07$ in all four cases. The plots of simulation results correspond to the average of 100 independent simulation runs. Exact solutions (solid curves) are obtained from Eq. (22).

giant components, the tails of the size distributions of the nongiant components in the case of $L > 0$ are heavier than their counterpart in the case of $L = 0$. This means that, for a fixed fraction of giant components, the sizes of the nongiant components diverge as L grows. In contrast, for Erdos-Renyi graphs with the same fraction of giant components, the tails of the size distributions of the nongiant components in the case of $L > 0$ are lighter than their counterpart in the case of $L = 0$. This means that for a fixed fraction of giant components, the sizes of the nongiant components diverge as L decreases. Precisely explaining this discrepancy is left for future work, which might lead to the characterization of differences between the two popular topologies from the new perspective. It is also worthwhile to note that the error in the case of $L = 1$ is more significant than its counterpart in the other cases, which hints that $L = 1$ is a very critical case (as we have already seen in Fig. 3). Analytically explaining this phenomenon is another interesting future work.

IV. CONCLUSION AND DISCUSSION

We introduced the notion of L-hop percolation, which is of conceptual value, because it captures a sort of *nonindependent* node deletion, and of practical value, because it leads to insights for cybersecurity (e.g., botnet defense). We presented analytic results on the L-hop percolation threshold as well as the mean size and size distribution of nongiant components of complex networks under such operations. In particular, we highlight the following findings: (a) Being able to trace to one-hop neighbors is "almost" as powerful as being able to trace to multihop neighbors, especially in power-law networks. (b) Most nongiant components in the case of Erdos-Renyi random graphs are isolated vertices, whereas typical nongiant components in the case of power-law graphs contain multiple edges. Moreover, as the number of chosen nodes increases, the giant component disappears abruptly in the cases of $L > 0$ but much more smoothly in the case of $L = 0$. (c) The tail of the distribution has a sharp difference between $L = 0$ and $L > 0$ for both Erdos-Renyi random graphs and power-law random graphs when the giant component fraction is kept as a constant.

The notion of L-hop percolation brings a range of interesting problems for future research. In addition to those mentioned in the text, here are more examples. What conclusions we can draw when considering other types of node deletion strategies? If we define some cost functions for node deletions corresponding to $L = 0$ and $L > 0$, respectively, how can we determine the optimal L value? In addition, one reviewer suggested the following interesting research problems: What is the degree distribution of the deleted nodes? Does it strongly differ from that found for $L = 0$? What happens on networks without such strong anticorrelations? What if the selection operations are targeted toward certain nodes? For the power-law degree distributions, is there a kind of power-law distribution when the occupation probability q is very close to the critical point q_c?

ACKNOWLEDGMENTS

We thank Mark Newman for answering our questions regarding our simulation study. We are grateful to the reviewers whose comments helped improve the paper significantly. This work was supported in part by AFOSR Grant No. FA9550-09-1-0165 and by an AFOSR MURI grant.

[1] M. Molloy and B. Reed, Random Struct. Algor. **6**, 161 (1995).
[2] M. Molloy and B. Reed, Comb. Probab. Comput. **7**, 295 (1998).
[3] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, Phys. Rev. E **64**, 026118 (2001).
[4] N. Fountoulakis, Internet Math. **4**, 329 (2007).
[5] S. Janson, Electron. J. Probab. **14**, 86 (2009).
[6] M. E. J. Newman, Phys. Rev. E **76**, 045101 (2007).
[7] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts, Phys. Rev. Lett. **85**, 5468 (2000).
[8] S. He, S. Li, and H. Ma, Physica A: Statistical Mechanics and Its Applications **388**, 4277 (2009).
[9] J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon, in *Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets* (USENIX Association, Berkeley, 2007), p. 1.
[10] B. Enright, G. Voelker, S. Savage, C. Kanich, and K. Levchenko, USENIX **33**, 2008.
[11] D. Dittrich and S. Dietrich, in *Proceedings of the 3rd International Conference On Malicious and Unwanted Software* (IEEE, Fairfax, 2008), p. 41.
[12] T. Holz, M. Steiner, F. Dahl, E. Biersacky, and F. Freiling, in *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats* (USENIX Association, Berkeley, 2008).
[13] G. Sinclair, C. Nunnery, and B. Kang, in *Proceedings of the 4th Annual Conference on Malicious and Unwanted Software* (IEEE, Montreal, 2009), p. 69.
[14] D. Dittrich, USENIX **34**, 35 (2009).
[15] B. Coskun, S. Dietrich, and N. Memon, in *Proceedings of the 26th Annual Computer Security Applications Conference* (ACM, Austin, 2010), p. 131.
[16] B. Stock, J. Göbel, M. Engelberth, F. Freiling, and T. Holz, in *Proceedings of the 2009 European Conference on Computer Network Defense* (IEEE, Washington, 2009), p. 13.
[17] J. Morales, R. Sandhu, and S. Xu, in *Proceedings of the 5th International Conference on Malicious and Unwanted Software* (IEEE, Nancy, 2010), p. 31.
[18] H. Wilf, *Generatingfunctionolgy*, 2nd ed. (Academic Press, London, 1994).
[19] B. Bollobas, *Random Graphs* (Cambridge University Press, New York, 2001).
[20] L. Ahlfors, *Complex Analysis* (McGraw-Hill, New York, 1979).
[21] M. Faloutsos, P. Faloutsos, and C. Faloutsos, in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (ACM, New York, 1999), p. 251.
[22] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes, Rev. Mod. Phys. **80**, 1275 (2008).
[23] M. Newman, SIAM Rev. **45**, 167 (2003).
[24] R. Cohen, S. Havlin, and D. ben-Avraham, Phys. Rev. Lett. **91**, 247901 (2003).

Push- and Pull-Based Epidemic Spreading in Networks: Thresholds and Deeper Insights

SHOUHUAI XU, University of Texas at San Antonio
WENLIAN LU, Fudan University
LI XU, University of Texas at San Antonio

Understanding the dynamics of computer virus (malware, worm) in cyberspace is an important problem that has attracted a fair amount of attention. Early investigations for this purpose adapted biological epidemic models, and thus inherited the so-called homogeneity assumption that each node is equally connected to others. Later studies relaxed this often unrealistic homogeneity assumption, but still focused on certain power-law networks. Recently, researchers investigated epidemic models in *arbitrary* networks (i.e., no restrictions on network topology). However, all these models only capture *push-based* infection, namely that an infectious node always actively attempts to infect its neighboring nodes. Very recently, the concept of *pull-based* infection was introduced but was not treated rigorously. Along this line of research, the present article investigates push- and pull-based epidemic spreading dynamics in arbitrary networks, using a nonlinear dynamical systems approach. The article advances the state-of-the-art as follows: (1) It presents a more general and powerful sufficient condition (also known as epidemic threshold in the literature) under which the spreading will become stable. (2) It gives both upper and lower bounds on the global mean infection rate, regardless of the stability of the spreading. (3) It offers insights into, among other things, the estimation of the global mean infection rate through localized monitoring of a small *constant* number of nodes, *without* knowing the values of the parameters.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: Epidemic dynamics, epidemic models, epidemic threshold, network, graph, eigenvalue

ACM Reference Format:
Xu, S., Lu, W., and Xu, L. 2012. Push- and pull-based epidemic spreading in networks: Thresholds and deeper insights. ACM Trans. Autonom. Adapt. Syst. 7, 3, Article 32 (September 2012), 26 pages.
DOI = 10.1145/2348832.2348835 http://doi.acm.org/10.1145/2348832.2348835

1. INTRODUCTION

Kephart and White [1991, 1993] initiated the study of computer virus dynamics by adapting certain homogeneous biological epidemic models [McKendrick 1926; Kermack

S. Xu and L. Xu were supported in part by grants sponsored by AFOSR, AFOSR MURI, ONR, and UTSA. W. Lu was supported in part by the Foundation for the Author of National Excellent Doctoral Dissertation of PR China no. 200921, the National Natural Sciences Foundation of China under grant no. 60804044, and the Shanghai Pujiang Program no. 08PJ14019.

Authors' addresses: S. Xu (corresponding author), Department of Computer Science, University of Texas at San Antonio, TX; email: shxu@cs.utsa.edu; W. Lu, Center for Computational Systems Biology and the School of Mathematical Sciences, Fudan University, PR China; L. Xu, Department of Computer Science, University of Texas at San Antonio, TX.

DOI 10.1145/2348832.2348835 http://doi.acm.org/10.1145/2348832.2348835

and McKendrick 1927; Bailey 1975; Anderson and May 1991; Hethcote 2000] where each individual has equal contact to the others. Recently, researchers (mainly statistical physicists) weakened the homogeneity assumption by considering spreading in heterogeneous networks, but mainly focused on spreading in power-law networks (see Moreno et al. [2002], Pastor-Satorras and Vespignani [2001, 2002], Newman [2003] and Barrat et al. [2008]). Very recently, computer scientists investigated the modeling of spreading dynamics in *arbitrary* networks (i.e., not just power-law ones) [Wang et al. 2003; Ganesh et al. 2005; Chakrabarti et al. 2008]. These models considered *push-based* spreading (attacks or infection), by which a virus always actively attempts to infect the susceptible computers. Push-based spreading (and thus these models) cannot accommodate *pull-based* attacks such as "drive-by download" [Provos et al. 2007] that a susceptible computer can get infected by connecting to a compromised Web site. This inspired researchers to introduce the more realistic *push-* and *pull-based* models in Li et al. [2007], which, however, conducted mainly a simulation-based study and did not offer deep insights. This calls for a systematic and analytic study of push- and pull-based epidemic spreading dynamics in arbitrary networks, which motivates the present article.

Although it is intuitive that push- and pull-based spreading dynamics models can accommodate attacks that are not captured by push-based models, one may still ask whether this is actually the case. This is a legitimate question because the abstraction of push-based spreading may be able to accommodate pull-based spreading as well (e.g., by attempting to adjust some parameters in push-based models). However, it turns out not to be the case because pull-based spreading and push-based spreading are two mechanisms that exhibit fundamentally/conceptually different behaviors. Specifically, pull-based spreading can accommodate the outside environment of a network in question (e.g., a compromised Web site does not belong to the network system in which the push-based epidemic spreading takes place), whereas push-based spreading cannot accommodate the outside environment. It therefore appears to be necessary to introduce a new type of parameter, called α, which abstracts the probability that a node gets infected because of its own reasons such as accessing a malicious Web site. As such, push-based spreading dynamics corresponds to the case of $\alpha = 0$, and push- and pull-based spreading dynamics corresponds to the case of $\alpha > 0$. In particular, when compared with the case of $\alpha = 0$, in the case of $\alpha > 0$ the spreading would never die out and the dynamics is much more difficult to analyze.

1.1. Our Contributions

First, we present a general sufficient condition (also known as epidemic threshold in the literature) under which the push- and pull-based epidemic spreading will become stable. This result supersedes its push-based counterpart given by Chakrabarti et al. [2008].[1] In other words, our sufficient condition is more general and powerful. We also give a more succinct variant sufficient condition which, among other things, allows to tune (in a quantitative rather than qualitative fashion) the model parameters so as to assure that the spreading will become stable. This has important implications. For example, we can first force the spreading dynamics to become stable and then use the global mean infection rate (i.e., the rate or portion of infected nodes) as an index to indicate the overall security of a network and to support higher-level decision making.

Second, we notice that it may not be always possible to force the spreading to become stable (e.g, the cost may be prohibitive). When the spreading is not stable (which does not necessarily mean there is an outbreak in the setting of this article with $\alpha > 0$), it

[1] As a side-product, we point out an error in Chakrabarti et al. [2008], by which their sufficient condition was mistakenly claimed to be "necessary" as well. The details will be given in Section 4.1.

would be important to know what are the worst-case and best-case scenarios in terms of the global mean infection rate. Towards this end, we give upper and lower bounds of the global mean infection rate. The bounds are actually applicable no matter the spreading dynamics is stable or not and are heuristically tight in some cases. Unfortunately, we are currently unable to precisely pin down the threshold above which the upper bound would be tight, and below which the lower bound would be tight. Still, we note that results related to nonstable scenarios are very difficult to obtain in general.

Third, we offer the following deeper insights that are, to our knowledge, the first of their kind when considering arbitrary networks. When the spreading is stable:

—we show how node degree governs the node infection rate;
—we give a condition under which the popular *mean field* approach is applicable to study push- and pull-based dynamics in arbitrary networks;
—we show how the global infection rate can be estimated by monitoring a small number of nodes selected in a certain fashion without knowing the values of the parameters and independent of the size of the network.

The rest of the article is organized as follows. In Section 2 we discuss the related prior works. Because our work is built on top of the concept/model of push- and pull-based spreading dynamics introduced in Li et al. [2007], we briefly review it in Section 3. In Section 4 we present some sufficient conditions (i.e., epidemic thresholds) under which the spreading will become stable, and present some upper and lower bounds on the global mean infection rate. In Section 5 we present some deeper insights that are especially useful in the absence of certain system information. We conclude the work in Section 6 with open questions for future research.

2. RELATED WORK

The study of epidemic spreading dynamics on complex networks has become a hot topic. From a computer science perspective, this perhaps has been catalyzed by the ubiquitous presence of real-life networks, including social networks, Internet, and email networks. The study has been inherently multidisciplinary, and researchers have investigated approaches such as epidemiology, dynamical systems, stochastic processes, percolation theory, generating functions, algebraic graph theory, and statistical physics. As such, a thorough and complete review of the literature is clearly beyond the scope of the present article. Instead, we here focus on a specific perspective, and classify the most relevant existing studies of computer virus/malware dynamics into the following three generations. (Because our focus is not about immunization, we do not aim to discuss the vast volume of literature in this subfield as well.)

The first generation, initiated by Kephart and White [1991, 1993] and followed by numerous variant studies such as Zou et al. [2002, 2003], focused on homogeneous networks. Such homogeneous models essentially require the underlying communications to form complete graphs. This is unrealistic because some computers (or IP addresses) are blocked from accessing certain other computers (or IP addresses), and because malware spreading does not necessarily exploit the fully random scanning strategy [Chen and Ji 2005]. Moreover, the spreading processes that exploit social networks (including email communication networks) are certainly not complete graphs, but rather follow the social networks' topologies.

The second generation focused on specific heterogenous (i.e., power-law) networks, for which we refer to the most recent excellent books [Barrat et al. 2008; Newman 2010] for a thorough treatment on the topic and for the large body of literature. From a technical perspective, we here highlight that there are mainly two approaches to tackling such heterogeneity. One approach is to exploit the specific properties of the degree distributions, especially the Barabasi-Albert power-law degree distribution and the affiliated preferential attachment model [Barabasi and Albert 1999], which, however, are not without limitations (see, for example, Faloutsos et al. [1999] and Willinger et al. [2009]). The large volume of research results in this widely known approach were nicely summarized in the 2008 book of Barrat et al. [2008]. The other (perhaps less known) approach is to exploit generating functions [Wilf 1994]. This approach has successfully made a connection to the percolation theory [Callaway et al. 2000; Newman et al. 2001; Newman 2007] and has successfully identified epidemic thresholds. Moreover, this approach is applicable to random graphs for a given degree sequence (which can correspond to power-law distributions) as introduced by Molloy and Reed [1995, 1998]. A systematic treatment of this approach can be found in the 2010 book of Newman [2010].

The third generation considers arbitrary heterogenous networks. The earlier investigations focused on push-based spreading [Wang et al. 2003; Ganesh et al. 2005; Chakrabarti et al. 2008]. These investigations made a clear connection between the fate of push-based dynamical processes in arbitrary networks and the algebraic properties of graphs (more specifically, the largest eigenvalue of the adjacency matrix). The present work enriches the dynamical processes by considering both push- and pull-based epidemic spreading, a concept first introduced in Li et al. [2007], which, however, did not go deep enough analytically. To be more precise, the two papers that are the predecessors to the present work are Chakrabarti et al. [2008], which presents the state-of-the-art epidemic threshold in push-based models, and Li et al. [2007], which introduced the push- and pull-based model but without giving deep results. We will demonstrate how our results supersede theirs.

Finally, we note that the epidemic spreading dynamics model considered in the present article is essentially the so-called Susceptible-Infection-Susceptible (SIS) dynamics in nature [Bailey 1975], which are very different from gossip processes (e.g., Demers et al. [1987], Karp et al. [2000], Kempe et al. [2001], Kempe and Kleinberg [2002], Shah [2009]), which are essentially Susceptible-Infection (SI) dynamics in nature.

3. A BRIEF REVIEW OF THE PUSH- AND PULL-BASED MODEL

We consider push- and pull-based epidemic spreading in complex networks, which are naturally represented as graphs. Because any topology could be possible and we want to draw insights that are widely applicable, we do not make any assumption on the networks' topologies, according to which the epidemic spreading takes place. In the rest of this section we review the push- and pull-based model introduced in Li et al. [2007].

Specifically, we consider a finite network graph $G = (V, E)$, where $V = \{1, 2, \dots, n\}$ is the set of nodes or vertices and E is the set of edges or arcs. Denote by $\mathsf{A} = [a_{vu}]$ the adjacency matrix of G, where $a_{vv} = 0$ and, if and only if $(u, v) \in E$ for $v \neq u$, we have $a_{vu} = 1$. The graph G captures the topology according to which the spreading takes place, where u can directly infect v only when $(u, v) \in E$. For undirected graph, let $\deg(v)$ denote the degree of node v in G. The discussion in the article focuses on undirected graphs, but can be adapted to the setting of directed graphs by replacing $\deg(v)$ as the in-degree of v in directed graph G.

Consider a discrete-time model with time $t = 0, 1, 2, \dots$. At any time t, a node $v \in V$ is either susceptible or infectious. (In this article, "infectious" and "infected" are used interchangeably.) Denote by $s_v(t)$ the probability that $v \in V$ is susceptible at time t, and $i_v(t)$ the probability that $v \in V$ is infectious at time t. The model invariant is $s_v(t) + i_v(t) = 1$ for any $t \geq 0$. Moreover, a susceptible node v may become infectious at a single time step because of push- or pull-based infection, and an infectious node may become susceptible because of defense or cure. This explains why our model is SIS (Susceptible-Infection-Susceptible) in nature, except that it considers both

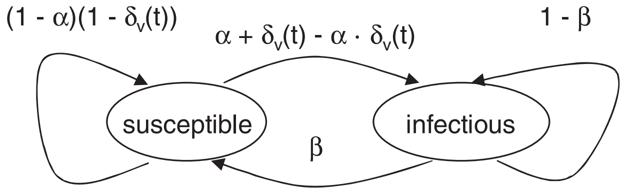


Fig. 1. The state transition diagram for node $v \in V$.

push- and pull-based infections. To accommodate push- and pull-based spreading, we use the following parameters.

—α: This is the pull-based infection capability, namely, the probability a susceptible node becomes infectious at a discrete time step because of its own activity (e.g., connecting to a malicious Web site which may not belong to G).
—γ: It is the push-based infection capability, namely, the probability that an infectious node u successfully infects a susceptible node v, where $(u, v) \in E$. In the case of undirected graph, it is natural that $\gamma_{uv} = \gamma_{vu}$ for all $(u, v) \in E$.
—β: This is the cure capability, namely, the probability that an infectious node becomes susceptible at a single time step.

The main notations used throughout the work are summarized next.

$\lambda_{1,\mathsf{A}}, \ldots, \lambda_{n,\mathsf{A}}$	the eigenvalues of the adjacency matrix A of network graph G with $\lambda_{1,\mathsf{A}} \geq \cdots \geq \lambda_{n,\mathsf{A}}$ (in modulus)
$\lambda_{\max}(M)$	the largest eigenvalue (in modulus) of matrix M
$M^\top$	the transpose of matrix (or vector) M
$s_v(t)$	the probability that node v is susceptible at time t
$i_v(t)$	the probability that node v is infectious at time t
$\bar{i}(t)$	the global mean infection rate (or probability) $\frac{1}{\lvert V \rvert} \sum_{v \in V} i_v(t)$; $\bar{i} = \lim_{t\to\infty} \bar{i}(t)$ is also used in the case the spreading is stable
$\bar{i}[t_0, t_1]$	the average of $\bar{i}(t)$ over time interval $t \in [t_0, t_1]$
$\langle r \rangle$	the mean of random variable r
α	the probability that a node becomes infectious at a single time step because of pull-based infection
β	the probability that an infectious node becomes susceptible at a single time step
γ	the probability that a susceptible node v is infected by an infectious neighbor u where $(u, v) \in E$ at a single time step
$\delta_v(t)$	the probability that a susceptible node v becomes infectious at time $t+1$ because of its infectious neighbors $\{u : (u, v) \in E\}$ at time t
I_n	the $n \times n$ identity matrix (i.e., $a_{jj'} = 1$ if $j = j'$, and $a_{jj'} = 0$ otherwise)

Figure 1 depicts the state transition diagram [Li et al. 2007], according to which node $v \in V$ changes its state. As in Chakrabarti et al. [2008], we may assume that the (neighboring) nodes' states are independent. Then, at time $t+1$, v may become infectious because of pull-based infection with probability α, or because of push-based infection with probability $\delta_v(t)$, where

$$\delta_v(t) = 1 - \prod_{(u,v)\in E} \left[1 - \gamma i_u(t)\right]. \tag{3.1}$$

As a result, the master equation of the nonlinear dynamical system is [Li et al. 2007]

$$\begin{cases} s_v(t+1) &= [(1-\alpha)(1-\delta_v(t))]s_v(t) + \beta i_v(t) \\ i_v(t+1) &= [1-(1-\alpha)(1-\delta_v(t))]s_v(t) + (1-\beta)i_v(t), \end{cases}$$

namely

$$i_v(t+1) = \left[1 - \left(1-\alpha\right) \prod_{(u,v)\in E} (1-\gamma i_u(t))\right)\right](1-i_v(t)) + (1-\beta)i_v(t). \tag{3.2}$$

Note that the preceding master equation preserves the invariant $s_v(t) + i_v(t) = 1$.

4. EPIDEMIC THRESHOLDS AND BOUNDS ON INFECTION RATE

In this section we address the following questions in the aforesaid push- and pull-based model.

—What are the sufficient conditions (i.e., epidemic thresholds) under which the spreading will become stable without resorting to any approximation (which is often used for analyzing nonlinear dynamical systems)? We address this question in Sections 4.1–4.2.

—Can we bound the node infection rate without requiring the spreading to become stable (Section 4.3)? Such results would be useful especially when it is impossible or prohibitive to "manipulate" or force the spreading dynamics to become stable.

In addition, we will discuss the applications of these theoretic results (Section 4.4).

4.1. A General Epidemic Threshold

Unlike the push-based model, where $\alpha = 0$ and thus $\lim_{t\to\infty} i_v(t) = 0$ for all v is a trivial equilibrium state, we do not have such a leverage for the push- and pull-based model because $\alpha > 0$ and thus it is almost always true that $i_v(t) > 0$, meaning that the spreading would not die out. Although we are unable to point out where the equilibrium state is, we show that there must be an equilibrium state (i.e., existence) that is exponentially globally stable.

THEOREM 4.1 (A GENERAL SUFFICIENT CONDITION UNDER WHICH THE SPREADING WILL BECOME STABLE). *Let $[i_1^*, \dots, i_n^*]$ be an equilibrium of nonlinear dynamical system (3.2), $H = diag[h_v]_{v=1}^n$ with*

$$h_v = \left| -\beta + (1-\alpha) \prod_{(u,v)\in E} (1-\gamma i_u^*) \right|$$

and the other parameters be specified as before. If

$$\lambda_{\max}(H + \gamma(1-\alpha)\mathsf{A}) < 1, \tag{4.1}$$

then system (3.2) is globally exponentially asymptotically stable, namely that $\lim_{t\to\infty} i_v(t) = i_v^*$ holds for $v = 1, \dots, n$ regardless of the number of the initially infected nodes.

PROOF. Let $I(t) = [i_1(t), \dots, i_n(t)]^\top$. Then Eq. (3.2) indicates $I(t+1) = f(I(t))$ for some continuous function $f : [0,1]^n \to [0,1]^n$. Since $[0,1]^n$ is convex, Brouwer's fixed point theorem [Yoshida 1971] says that there exists at least one equilibrium $i^* = [i_1^*, \dots, i_n^*]^\top$, that is,

$$i_v^* = \left[1 - (1-\alpha)\prod_{(u,v)\in E}(1-\gamma i_u^*)\right](1-i_v^*) + (1-\beta)i_v^*,\ \forall\, v \in V.$$

Let $z_v(t) = i_v(t) - i_v^*$ for all $v = 1, \ldots, n$. Therefore, we have

$$\begin{aligned} z_v(t+1) \;=\; & -\beta z_v(t) + (1-\alpha)\prod_{(u,v)\in E}(1-\gamma i_u^*)z_v(t) \\ & + (1-\alpha)\left[\prod_{(u,v)\in E}(1-\gamma i_u^*) - \prod_{(u,v)\in E}(1-\gamma i_u(t))\right](1-i_v(t)). \end{aligned}$$

Note that for each $v = 1, \ldots, n$,

$$\begin{aligned} & \prod_{(u,v)\in E}(1-\gamma i_u^*) - \prod_{(u,v)\in E}(1-\gamma i_u(t)) \\ & = \gamma \sum_{(u',v)\in E} z_{u'}(t)\cdot \prod_{(u,v)\in E,\ u<u'}(1-\gamma i_u^*)\cdot \prod_{(k,v)\in E,\ k>u'}(1-\gamma i_k(t)), \end{aligned}$$

we have

$$\begin{aligned} |z_v(t+1)| \;\le\; & h_v|z_v(t)| \\ & + (1-\alpha)\gamma \sum_{(u,v)\in E}|z_u(t)|\left|\prod_{(u,v)\in E,\ u<u'}(1-\gamma i_u^*)\prod_{(k,v)\in E,\ k>u'}(1-\gamma i_k(t))\right| \\ \le\; & h_v|z_v(t)| + (1-\alpha)\gamma\sum_{(u,v)\in E}|z_u(t)|. \end{aligned}$$

Define the following comparison system

$$\theta_v(t+1) = h_v\theta_v(t) + (1-\alpha)\gamma\sum_{(u,v)\in E}\theta_u(t) \tag{4.2}$$

with initial values $\theta(0)_v = |z_v(0)| \ge 0$ for $v = 1, \ldots, n$. This implies that $|z_v(t)| \le \theta_v(t)$ holds for all $v = 1, \ldots, n$ and $t \ge 0$. Let $\Theta(t) = [\theta_1(t), \ldots, \theta_n(t)]^\top$. Then Eq. (4.2) has the following matrix form

$$\Theta(t+1) = [H + \gamma(1-\alpha)\mathsf{A}]\Theta(t), \tag{4.3}$$

where A is the adjacency matrix of the network graph. Since the graph is connected, the matrix $\mathsf{X} = H + \gamma(1-\alpha)\mathsf{A}$ is irreducible and nonnegative. The Perron-Frobenius theorem [Horn and Johnson 1985] says that the spectral radius of X is its largest eigenvalue in modulus. Therefore, condition (4.1) implies that all eigenvalues of the matrix X are less than 1 in modulus. This implies that system (4.2) converges to zero exponentially. Because $|z_v(t)| \le \theta_v(t)$, $\lim_{t\to\infty}|z_v(t)| = 0$ holds for all $v = 1, \ldots, n$, that is, $\lim_{t\to\infty} i_v(t) = i_v^*$ holds for all $v = 1, \ldots, n$ and the convergence is at an exponential pace. This completes the proof. □

The previous theorem says that when the sufficient condition is satisfied, the spreading will converge to the equilibrium state (but not dying out) at an exponential pace, regardless of the number of initially infected nodes.

Comparing the sufficient condition presented in Chakrabarti et al. [2008] to ours. Our result supersedes the sufficient condition for the case of $\alpha = 0$ presented in Chakrabarti et al. [2008] because of the following corollary of Theorem 4.1.

COROLLARY 4.2. *In the case $\alpha = 0$, if*

$$\lambda_{1,\mathsf{A}} < \frac{\beta}{\gamma}, \tag{4.4}$$

the spreading dies out regardless of the number of initially infected nodes, that is, $\lim_{t\to\infty} i_v(t) = 0$ for all $v = 1, \ldots, n$.

PROOF. Note that $\alpha = 0$ and $i_v^* = 0$ implies $H = (1-\beta)I_n$ in Theorem 4.1. So, condition (4.1) becomes $\lambda_{\max}[(1-\beta)I_n + \gamma A] < 1$, which is equivalent to $1 - \beta + \gamma\lambda_{\max}(A) < 1$, namely condition (4.4). Hence, the corollary holds. □

While Corollary 4.2 corresponds to the sufficient condition given in Chakrabarti et al. [2008], meaning that our sufficient condition is strictly more powerful, the sufficient condition for the case of $\alpha = 0$ is not compatible with the more succinct sufficient condition for the case of $\alpha > 0$ (presented in Section 4.2). This also illustrates why the treatment in the case of $\alpha > 0$ is different.

Caveat: The conditions are sufficient *but not* necessary. Chakrabarti et al. [2008] claimed through their Theorem 2 that $\tau = \frac{1}{\lambda_{1,\mathsf{A}}} > \frac{\beta}{\delta}$ (in their terminology), which is equivalent to the preceding condition (4.4), is also the *necessary* condition for the spreading to die out for the case of $\alpha = 0$. Now we point out that their proof is flawed and explain why. We use their terminology so that it is easier to check against their proof, which is presented in Appendix A of Chakrabarti et al. [2008]. Their proof strategy is to prove "if the spreading dies out, then $\tau > \frac{\beta}{\delta}$." As shown in the derivation of their Eq. (A6), this requires to prove "if the system is asymptotically stable, then $\lambda_{i,\mathsf{S}} = 1 - \delta + \beta\lambda_{i,\mathsf{A}} < 1$ for $\forall i$, where $\mathsf{S} = \nabla g(\vec{0}) = \beta\mathsf{A} + (\vec{1} - \delta)\mathsf{I}$." This subsequently requires to prove "if the system is asymptotically stable at $\vec{P} = \vec{0}$, then the eigenvalues of $\nabla g(\vec{0})$ are less than 1 in absolute value." The proof of this claim was attributed to their Lemma 1, which however—as we now point out—cannot get through because the lemma actually states the opposite, namely "if the eigenvalues of $\nabla g(\vec{0})$ are less than 1 in absolute value, then the system is asymptotically stable at $\vec{P} = \vec{0}$." Therefore, the necessity proof in Chakrabarti et al. [2008] is flawed.

Having pointed out the flaw, one would wonder whether it is possible to prove "if the system is asymptotically stable at $\vec{P} = \vec{0}$, then the eigenvalues of $\nabla g(\vec{0})$ are less than 1 in absolute value" or "$\tau > \frac{\beta}{\delta}$ in Chakrabarti et al. [2008] is indeed the necessary condition for the spreading to die out." It turns out not to be the case. Conceptually, for a nonlinear dynamical system, when the largest eigenvalue of the Jacobin matrix at equilibrium equals to one exactly, it may not imply that the system is not asymptotically stable. For example, consider a simple system

$$x(t+1) = x(t) - \gamma(x(t))^2 \tag{4.5}$$

with $0 < \gamma < 1/2$ and $x(0) \in [0, 1]$. Since $x(t)$ monotonically decreases and belongs to the interval $[0, 1]$ for all t, it must converge to a fixed point of the system. Since zero is the only fixed point of system (4.5), we conclude that $\lim_{t\to\infty} x(t) = 0$. Even though the modulus of the derivative of the right-hand side at zero is 1, the system converges to zero if the initial value is less than 1. In other words, their condition (as well as ours) is sufficient but not necessary.

Having pointed out that their proof strategy for necessity cannot get through, one may wonder whether the necessity can be proven with a different strategy. It turns out not to be the case, as we show here with a concrete counter-example that a sufficient condition is not necessary in general. Our counter-example that follows shows that (in our terminology) there exist graphs where the infection rate can still go to zero when

$\lambda_{1,A} = \frac{\beta}{\gamma}$, which immediately violates the necessity of the sufficient condition. Let us consider a graph with only two nodes linked with a single edge. Then, we have the following dynamical system to describe the infection rate.

$$\begin{cases} i_1(t+1) = \gamma i_2(t)[1 - i_1(t)] + (1-\beta)i_1(t) \\ i_2(t+1) = \gamma i_1(t)[1 - i_2(t)] + (1-\beta)i_2(t) \end{cases} \tag{4.6}$$

In this case, the adjacency matrix is $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, whose largest eigenvalue in modulus is $\lambda_{1,A} = 1$. So, under the condition $\lambda_{1,A} = \frac{\beta}{\gamma}$, we have $\beta = \gamma$. Suppose $0 < \gamma < 1/2$. Then, $0 \le i_2(t) \le 1$ implies $1 - \gamma - \gamma i_2(t) > 0$. Let $c \in [0, 1]$ such that $i_1(t) \le c$ and $i_2(t) \le c$ at time t, we have $\gamma - c\gamma \ge 0$ and thus

$$\begin{aligned} i_1(t+1) &= [1-\gamma-\gamma i_2(t)]i_1(t) + \gamma i_2(t) \le [1-\gamma-\gamma i_2(t)]c + \gamma i_2(t) \\ &= i_2(t)(\gamma - \gamma c) + c(1-\gamma) \le c(\gamma - \gamma c) + c(1-\gamma) = c - c^2\gamma. \end{aligned}$$

In a similar fashion we have $i_2(t+1) \le c - c^2\gamma$. Still considering the comparison system (4.5) with $x(0) = 1$, we observe that $i_1(t) \le x(t)$ and $i_2(t) \le x(t)$ hold. From the preceding reasoning, we have $\lim_{t\to\infty} x(t) = 0$. This implies that $\lim_{t\to\infty} i_1(t) = 0$ and $\lim_{t\to\infty} i_2(t) = 0$, which serves as a desired counter-example.

4.2. A More Succinct Epidemic Threshold

The previous sufficient condition is general but not succinct. This motivates us to present the following less general, but more succinct, sufficient condition.

THEOREM 4.3 (A MORE SUCCINCT SUFFICIENT CONDITION UNDER WHICH THE SPREADING WILL BECOME STABLE). *Let* $m = \max_{v\in V} \deg(v)$. *In the case* $\beta < (1-\alpha)(1+(1-\gamma)^m)/2$, *if*

$$\lambda_{1,\mathsf{A}} < \frac{\alpha+\beta}{\gamma(1-\alpha)},$$

then the spreading will become stable regardless of the initial number of infected nodes.

In the case $\beta \ge (1-\alpha)(1+(1-\gamma)^m)/2$, *if*

$$\lambda_{1,\mathsf{A}} < \frac{1-\beta+(1-\alpha)(1-\gamma)^m}{\gamma(1-\alpha)},$$

then the spreading will become stable regardless of the initial number of infected nodes.

PROOF. Let $\psi = \max(|1-\alpha-\beta|, |-\beta+(1-\alpha)(1-\gamma)^m|)$, and the other parameters be specified as earlier. Because

$$(1-\alpha)(1-\gamma)^m \le (1-\alpha) \prod_{(u,v)\in E} (1-\gamma i_u^*) \le (1-\alpha),$$

we have

$$h_v \le \max\{|1-\alpha-\beta|, |-\beta+(1-\alpha)(1-\gamma)^m|\}.$$

So, the largest eigenvalue of $H + (1-\alpha)\gamma\mathsf{A}$ is smaller than that of $\psi I_n + (1-\alpha)\gamma\mathsf{A}$. Note that if

$$\psi + \gamma(1-\alpha)\lambda_{1,\mathsf{A}} < 1, \text{ or } \lambda_{1,\mathsf{A}} < \frac{1-\psi}{\gamma(1-\alpha)}, \tag{4.7}$$

then $\lambda_{\max}[H + (1-\alpha)\gamma\mathsf{A}] < 1$. This means, by applying Theorem 4.1, that the spreading will become stable regardless of the number of initially infected nodes.

Observe that in the case $\beta < (1-\alpha)(1+(1-\gamma)^m)/2$, sufficient condition (4.7) becomes

$$\lambda_{1,\mathsf{A}} < \frac{\alpha+\beta}{\gamma(1-\alpha)}.$$

In the case $\beta \geq (1-\alpha)(1+(1-\gamma)^m)/2$, sufficient condition (4.7) becomes

$$\lambda_{1,\mathsf{A}} < \frac{1-\beta+(1-\alpha)(1-\gamma)^m}{\gamma(1-\alpha)}.$$

This completes the proof. □

Comparing the sufficient condition given in Li et al. [2007] to ours. Li et al. [2007] also presented a sufficient condition for the case $\alpha > 0$, namely

$$\lambda_{1,\mathsf{A}} < \frac{\alpha+\beta}{\gamma} \text{ and } \lambda_{n,\mathsf{A}} > \frac{\alpha+\beta-2}{\gamma},$$

where $\lambda_{n,\mathsf{A}}$ is the smallest (in modulus) eigenvalue of A. However, in the derivation of their sufficient condition they used the following approximation of Eq. (3.2)

$$i_v(t+1) \approx \alpha + (1-\alpha-\beta)i_v(t) + \gamma \sum_{(u,v)\in E} i_u(t),$$

which omits all the nonlinear terms and is therefore quite coarse.

In contrast, our sufficient condition given in Theorem 4.3 is derived *without* using any approximation. As a result, our sufficient condition is advantageous because it is both *more concise* and *weaker*. Our sufficient condition is more concise because it says that $\lambda_{n,\mathsf{A}}$ is irrelevant, whereas their sufficient condition unnecessarily involves $\lambda_{n,\mathsf{A}}$. Our sufficient condition is weaker because when $\beta < (1-\alpha)(1+(1-\gamma)^m)/2$, $\lambda_{1,\mathsf{A}} < \frac{\alpha+\beta}{\gamma}$ implies $\lambda_{1,\mathsf{A}} < \frac{\alpha+\beta}{\gamma(1-\alpha)}$ but the latter does not imply the former.

4.3. Bounding the Node Infection Rate

The aforesaid theorems state sufficient conditions under which the spreading will become stable. Because we are unable to compute the equilibrium state for $\alpha > 0$, the next natural question is: Can we bound the nodes' infection rates? Now we address this question.

THEOREM 4.4 (BOUNDS OF $i_v(t)$ REGARDLESS OF THE STABILITY OF THE SPREADING). *Let $\overline{\lim}_{t\to\infty} i_v(t)$ denote the upper bound of the limit of $i_v(t)$, and $\underline{\lim}_{t\to\infty} i_v(t)$ denote the lower bound of the limit of $i_v(t)$. Then, we have*

$$\overline{\lim}_{t\to\infty} i_v(t) \leq \theta_v^+ \text{ and } \underline{\lim}_{t\to\infty} i_v(t) \geq \theta_v^-,$$

where

$$\theta_v^+ = \frac{1-(1-\alpha)(1-\gamma)^{\deg(v)}}{\min\{1+\beta-(1-\alpha)(1-\gamma)^{\deg(v)}, 1\}},$$

$$\theta_v^- = \begin{cases} \frac{1-(1-\alpha)(1-\gamma\nu)^{\deg(v)}}{1+\beta-(1-\alpha)(1-\gamma\nu)^{\deg(v)}} & (1-\alpha)(1-\gamma\nu)^{\deg(v)} \geq \beta \\ ((1-\alpha)(1-\gamma\nu)^{\deg(v)} - \beta)\theta_v^+ + 1 - (1-\alpha)(1-\gamma\nu)^{\deg(v)} & \textit{otherwise} \end{cases},$$

with $\nu = \min\{1-\beta, \alpha\}$.

PROOF. First, consider the upper bound. Because

$$i_v(t+1) \leq [1-(1-\alpha)(1-\gamma)^{\deg(v)}](1-i_v(t)) + (1-\beta)i_v(t),$$

we have

$$i_v(t+1) - \theta_v^+ \leq [(1-\alpha)(1-\gamma)^{\deg(v)} - \beta](i_v(t) - \theta_v^+).$$

In the case $(1-\alpha)(1-\gamma)^{\deg(v)} \geq \beta \geq 0$, there are two scenarios. If there exists some t_0 such that $i_v(t_0) < \theta_v^+$, then we have $i_v(t+1) \leq \theta_v^+$ for all $t \geq t_0$. If $i_v(t) \geq \theta_v^+$ for all t, then

$$|i_v(t+1) - \theta_v^+| \leq |[(1-\alpha)(1-\gamma)^{\deg(v)} - \beta]||i_v(t) - \theta_v^+|.$$

Because $|(1-\alpha)(1-\gamma)^{\deg(v)} - \beta| < 1$, we have

$$\lim_{t\to\infty}(i_v(t) - \theta_v^+) = 0.$$

So, we have $\overline{\lim}_{t\to\infty} i_v(t) \leq \theta_v^+$ in both scenarios.

In the case $(1-\alpha)(1-\gamma)^{\deg(v)} < \beta$, we immediately have

$$\begin{aligned} i_v(t+1) &\leq [(1-\alpha)(1-\gamma)^{\deg(v)} - \beta]i_v(t) + 1 - (1-\alpha)(1-\gamma)^{\deg(v)} \\ &\leq 1 - (1-\alpha)(1-\gamma)^{\deg(v)}. \end{aligned}$$

Therefore, in both cases, we have $\overline{\lim}_{t\to\infty} i_v(t) \leq \theta_v^+$.

Second, consider the lower bound. Since $1 - (1-\alpha)(1-\delta_v(t)) \geq \alpha$, we can conclude that $i_v(t+1) \geq \alpha(1 - i_v(t)) + (1-\beta)i_v(t)$. The fact $i_v(t) \in [0, 1]$ implies that the previous right-hand side is greater than ν. So, we have $i_v(t) \geq \nu$. Thus, we have

$$i_v(t+1) \geq [1 - (1-\alpha)(1-\gamma\nu)^{\deg(v)}](1 - i_v(t)) + (1-\beta)i_v(t).$$

In the case $(1-\alpha)(1-\gamma\nu)^{\deg(v)} \geq \beta$, we can rewrite the preceding inequality as

$$i_v(t+1) - \theta_v^- \geq [(1-\alpha)(1-\gamma\nu)^{\deg(v)} - \beta](i_v(t) - \theta_v^-).$$

We observe that there are two scenarios. If there exists some t_1 such that $i_v(t_1) > \theta_v^-$, then $i_v(t) \geq \theta_v^-$ holds for all $t \geq t_1$. If $i_v(t) \leq \theta_v^-$ holds for all $t \geq 0$, then

$$|i_v(t+1) - \theta_v^-| \leq |[(1-\alpha)(1-\gamma\nu)^{\deg(v)} - \beta]||i_v(t) - \theta_v^-|.$$

Because $|[(1-\alpha)(1-\gamma\nu)^{\deg(v)} - \beta]| < 1$, we have $\lim_{t\to\infty} |i_v(t+1) - \theta_v^-| = 0$. Therefore, we have $\underline{\lim}_{t\to\infty} i_v(t) \geq \theta_v^-$ in both scenarios.

In the case $(1-\alpha)(1-\gamma)^{\deg(v)} < \beta$, we immediately have

$$\begin{aligned} &\underline{\lim}_{t\to\infty} i_v(t+1) \\ &\geq \underline{\lim}_{t\to\infty}\big\{\big[(1-\alpha)(1-\gamma\nu)^{\deg(v)} - \beta\big]i_v(t)) + 1 - (1-\alpha)(1-\gamma\nu)^{\deg(v)}\big\} \\ &\geq [(1-\alpha)(1-\gamma\nu)^{\deg(v)} - \beta]\overline{\lim}_{t\to\infty} i_v(t)) + 1 - (1-\alpha)(1-\gamma\nu)^{\deg(v)} \\ &\geq [(1-\alpha)(1-\gamma\nu)^{\deg(v)} - \beta]\theta_v^+ + 1 - (1-\alpha)(1-\gamma\nu)^{\deg(v)}. \end{aligned}$$

Therefore, in both cases, we have $\underline{\lim}_{t\to\infty} i_v(t) \geq \theta_v^-$. This completes the proof. □

The inequalities we use to prove Theorem 4.4 indicate that when the actual infection rate is significantly smaller than 1, the lower bound would be tighter; when the actual infection rate is close to 1, the upper bound would be tighter. This heuristics is confirmed by our simulation results. However, it is challenging to precisely pin down the infection rate threshold, above which the upper bound is tight and below which the lower bound is tight. We leave this to future research, while pointing out that a key difficulty comes from the fact hat the spreading may not be stable.

4.4. Applications of the Thresholds and Bounds

In addition to their theoretic significance, the thresholds and bounds have good applications, especially for quantitatively guiding the operation of tuning the parameters so as to achieve the desired outcomes.

First, the thresholds can be used to guide the adjustment of parameters so as to make the spreading stable. For this purpose, it is intuitive to decrease γ (e.g., by imposing a more thorough examination on message packets) and/or α (e.g., by blocking access to potentially malicious Web sites), and/or increase β (e.g., by deploying more powerful, and thus likely more expensive, defense tools). It would be less intuitive that the defender can also seek to decrease $\lambda_{1,\mathsf{A}}$ by deleting edges and/or nodes. It would be far less intuitive that the optimal strategy is to delete edges/nodes so as to reduce $\lambda_{1,\mathsf{A}}$ as much as possible, rather than to delete (for example) the largest-degree nodes as noted in Chakrabarti et al. [2008]. In any case, how much adjustment is enough to assure that the spreading will become stable? The thresholds (more specifically, the inequalities) offered in Theorems 4.1 and 4.3 can be used to answer such questions quantitatively. Moreover, if the costs incurred due to the adjustment of parameters are known, then Theorems 4.1 and 4.3 also provide a basis for cost-effectively adjusting the parameters.

Second, consider the case of $\alpha = 0$. Suppose it is impossible or over-costly to make the spreading die out (e.g., because the parameters α, β, γ, and $\lambda_{1,\mathsf{A}}$ cannot be made arbitrarily small/large, while assuring that the network remains functioning). This means that the conditions given in Theorems 4.1 and 4.3 are not satisfied any more. Even if the spreading might not be stable, we can still utilize the bounds given in Theorem 4.4 to achieve some useful goals. On one hand, the upper bound can be used to estimate the worst-case infection rate and, perhaps more importantly, can be used to guide the tuning of parameters in the bound θ_v^+ so as to reduce it as much as possible. Although it is not necessarily a tight estimation of the infection rate, it still would be useful because we now can control the worst-case infection rate under a desired level. On the other hand, the lower bound can be used to estimate in a sense the best-case scenario. Putting the bounds together, we further get a clue on how tight/loose the bounds are and, when the two bounds are close to each other, what is the small interval in which the infection rate resides. Finally, assuming the relevant cost functions are known, the tuning of parameters can also be made cost effective. Such quantitative guidance would be always preferred to the qualitative intuitions.

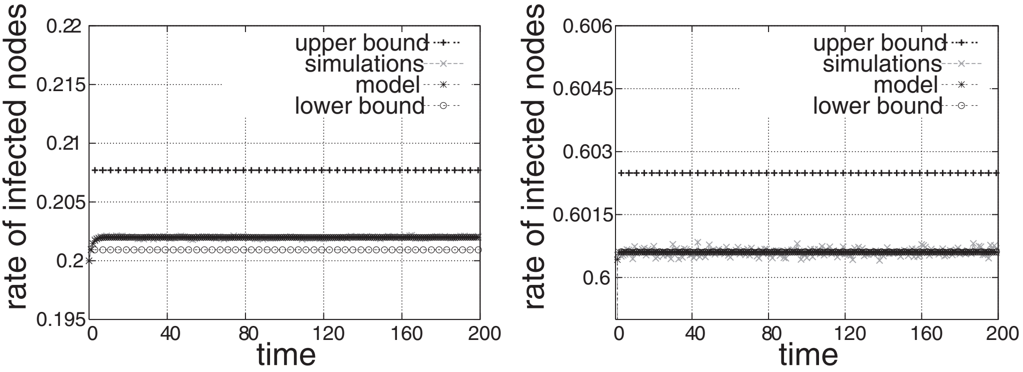
Third, consider the case of $\alpha > 0$, which means that the spreading would never die out. In this case, even if we cannot make the spreading stable (similarly because it is either impossible or over-costly in practice as in the case of $\alpha = 0$), the previous discussion on utilizing Theorem 4.4 to guide the adjustment of parameters would be still applicable here. Moreover, as shown in Section 5.1, the bounds hinted us to conduct some deeper analysis that leads to useful insights.

4.5. Confirming the Thresholds and Examining the Bounds via Simulation

We conducted a simulation study to confirm the analytical results by using two real-life complex network datasets obtained from http://snap.stanford.edu/data/.

—Epinions online social network dataset. This is a directed graph corresponding to a real-life online social network, where nodes represent people and arcs represent interactions between them. The graph has 75,879 nodes and 508,837 arcs with average node in- and out-degree 6.7059, maximal node in-degree 3,035, maximum node out-degree 1,801, and $\lambda_{1,\mathsf{A}} = 106.53$.

—Enron email dataset. This is a graph representing Enron's internal email communications, where nodes represent employees and arcs represent email messages. Unlike



(a) $\alpha = 0.1$ (y-axis resolution: 10^{-3})

(b) $\alpha = 0.6$ (y-axis resolution: 10^{-4})

Fig. 2. The case of Epinions dataset ($\beta = 0.1$ and $\gamma = 0.004$).

the Epinions dataset, $(u, v) \in E$ always implies $(v, u) \in E$ in Enron dataset (that is, the interactions between a pair of nodes, if any, are always mutual), which allows us to treat it as an undirected graph. The graph has 36,692 nodes and 367,662 edges with average degree 20.0404, maximal node degree 2,766, and $\lambda_{1,A} = 118.4177$.

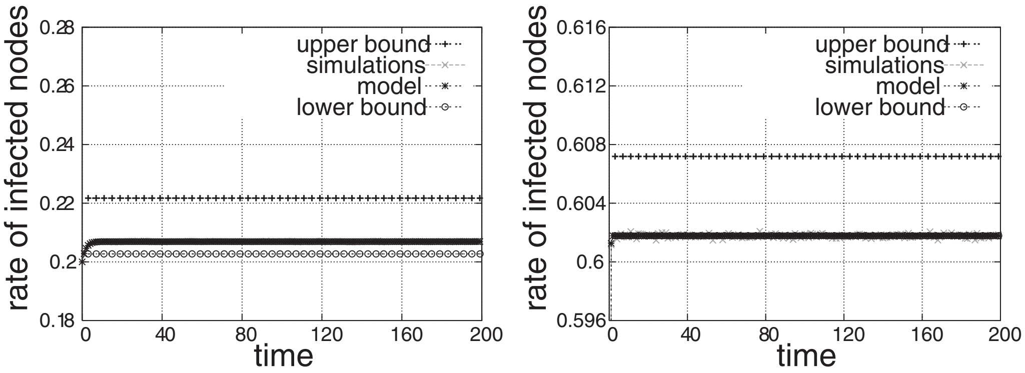
Although not reported here, we note that both networks exhibit power-law degree distributions. The reason of using these datasets is that push- and pull-based epidemic spreading could take place in such networks.

Using the preceding datasets, we compare the simulation results (averaged over 500 independent runs) of the global mean infection rate $\bar{i}(t) = \frac{1}{|V|}\sum_{v \in V} i_v(t)$; the average (over all nodes) of the results obtained from Eq. (3.2); the average (over all nodes) of the bounds given in Theorem 4.4. The parameters are selected according to Theorem 4.3 so that the spreading will become stable. In all cases, we let 20% randomly picked nodes be initially infected. For the sake of better visual effect, we use higher resolutions for plotting figures whenever possible.

Using the Epinions network dataset, Figure 2 plots the comparison of two parameter scenarios; in both cases the spreading will become stable. First, we note that the simulation result still oscillates a little bit in Figure 2(b) because the y-axis resolution is very high (10^{-4}). Indeed, we calculated the standard deviations of the simulation results in both scenarios, and found that the standard deviations are not visually noticeable (which explains why we did not plot the standard deviations). We interpret (at least partially) this phenomenon as follows. Since the initial conditions are random, the asymptotic limit of the infection rate, i_v^*, supposing it exists, is a random variable. Hence, the variance of the mean infection rate $\bar{i} = \frac{1}{n}\sum_{v=1}^{n} i_v^*$ is

$$var(\bar{i}) = \frac{1}{n^2} var\left(\sum_{v=1}^{n} i_v^*\right) \leq \frac{2}{n^2}\sum_{v=1}^{n} var(i_v^*).$$

If $var(i_v^*)$ is bounded from above by some positive constant, then $var(\bar{i})$ converges to zero as $n \to \infty$, namely as the size of the network goes to infinity. Hence, the standard deviation, $\sqrt{var(\bar{i})}$, converges to zero. This would at least partially interpret why the observed standard deviations are very small. Second, even if in Figure 2(a) with $\alpha = 0.1$, we observe that the lower bounds are relatively tight. This in a sense confirms the aforementioned heuristics that when the infection rate is far from 1, the lower bound



(a) $\alpha = 0.1$ (y-axis resolution: 10^{-2}) (b) $\alpha = 0.6$ (y-axis resolution: 10^{-3})

Fig. 3. The case of Enron email dataset ($\beta = 0.4$ and $\gamma = 0.001$).

would be relatively tight. Third, it is noted that the simulation results and the model predictions are almost always lumped together.

Similar phenomena are observed in the case of Enron email dataset (Figure 3).

5. DEEPER INSIGHTS

In this section, we draw deeper insights related to the following three questions when the spreading is stable.

—Does deg(v) govern $i_v^* = \lim_{t\to\infty} i_v(t)$, namely what role do node degrees play in terms of determining the nodes' infection rates (Section 5.1)?

—Under what conditions the mean field approach is reasonably accurate and thus applicable to arbitrary networks (Section 5.2)?

—When the mean field approach is reasonably accurate, how can we exploit it to infer the global mean infection rate through localized monitoring (Section 5.3)? This is especially important because it does not require knowledge of the parameters.

5.1. The Role of Node Degree on Node Infection Rate

Both the upper and lower bounds given in Theorem 4.4 are a function of deg(v). This suggests as a special case that i_v^*, the probability that node v is infected when the spreading becomes stable, may be dependent upon deg(v). To characterize the impact of deg(v) on i_v^*, we start with an observation. Since we want to reveal the relation between i_v^* and deg(v), we treat it as if i_v^* depends only on deg(v). This allows us to consider $i^*(k)$, the probability any degree-k node is infected after the spreading becomes stable. Suppose, for feasibility, all of v's neighboring nodes have identical infection rate as v, namely that $i_u^* \approx i_v^*$ for all u with $(u, v) \in E$. From master equation (3.2), we have

$$\begin{aligned} i_v^* &= \left[1-(1-\alpha)\prod_{(u,v)\in E}(1-\gamma i_u^*)\right](1-i_v^*)+(1-\beta)i_v^* \\ &\approx [1-(1-\alpha)(1-\gamma i_v^*)^k](1-i_v^*)+(1-\beta)i_v^*. \end{aligned} \tag{5.1}$$

Hence, solution to the following equation

$$\beta i^*(k) = [1-(1-\alpha)(1-\gamma i^*(k))^k](1-i^*(k)) \tag{5.2}$$

is the probability that any degree-k node is infected. In other words, the solution to Eq. (5.3) approximates $i^*(k)$. We have

$$\beta x = [1 - (1-\alpha)(1-\gamma x)^k](1-x) \tag{5.3}$$

for each $k = 1, 2, \ldots, \max_{v \in V}\{\deg(v)\}$. We observe that Eq. (5.3) can be numerically solved using (for example) Matlab.

In order to evaluate the preceding approximation-based analytic heuristics, we conducted a simulation study. In order to attest the applicability of the heuristics, we further used, in addition to the aforementioned Epinions and Enron datasets, the following datasets which vary in network topology and/or size.

—Oregon autonomous system dataset. This is an undirected graph of real-life network connections between Internet autonomous systems. The dataset is obtained from Web site http://topology.eecs.umich.edu/data.html. The graph has 11,461 nodes and 32,730 edges with average node degree 5.7115, maximal node degree 2,432, and largest eigenvalue $\lambda_{1,\mathsf{A}} = 75.2407$. It also exhibits a power-law degree distribution.

—Power-law graphs. We used the Brite tool [Medina et al. 2001] to generate two synthetic undirected power-law graphs. The small one has 2,000 nodes and 5,997 edges with average node degree 5.997, maximal node degree 90, and $\lambda_{1,\mathsf{A}} = 11.6514$. The large one has 37,000 nodes and 369,908 edges with average node degree 19.995, maximal node degree 1,530, and $\lambda_1 = 103.9362$.

—Random graphs. We used two synthetic undirected Erdos-Renyi graphs. The small one has 2,000 nodes and 6,001 edges with average node degree 6.001, maximal node degree 16, and $\lambda_{1,\mathsf{A}} = 7.1424$. The large one has 37,000 nodes and 370,000 edges with average node degree 20, maximal node degree 40, and $\lambda_1 = 21.0894$.

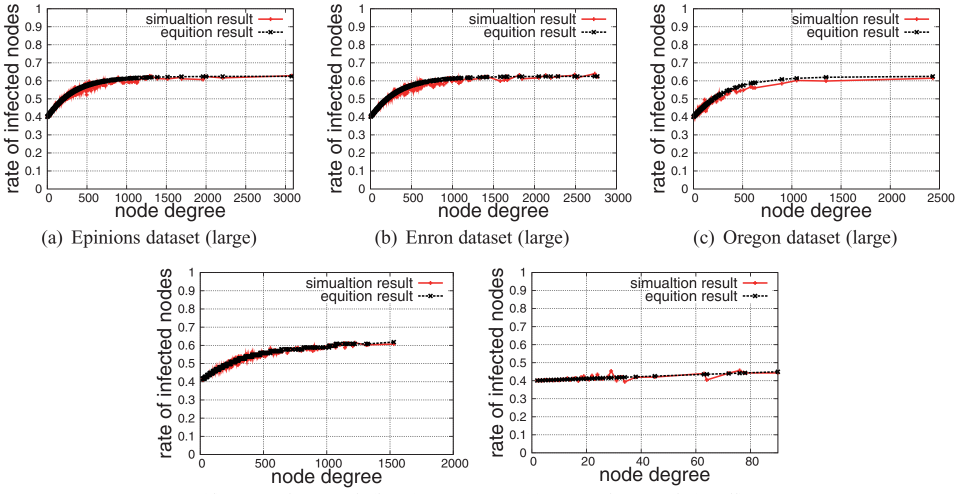
—Regular graphs. The small synthetic undirected regular graph has 2,000 nodes and 6,000 edges with node degree 6 and $\lambda_{1,\mathsf{A}} = 6$. The large synthetic undirected regular graph has 37,000 nodes and 370,000 edges with node degree 20 and $\lambda_1 = 20$.

In our simulations, we set $\alpha = 0.4$, $\beta = 0.6$ and $\gamma = 0.004$, which always satisfy the sufficient condition given in Theorem 4.3 and thus the spreading will become stable. We compare in Figures 4 and 5 the $i^*(k)$ obtained from simulation and the $i^*(k)$ obtained by solving Eq. (5.3). In each case, the x-axis represents degree $k = 1, \ldots, \max\{\deg(v)\}$, and the y-axis represents simulation-based $i^*(k)$ (+ in red color), which is obtained by averaging (over 100 simulation runs) the i_v^*'s of the degree-k nodes after the spreading becomes stable, and the $i^*(k)$ obtained by numerically solving Eq. (5.3) (× in black color).

In the case of both real and synthetic power-law graphs (refer to Figure 4) the numerical results fit the simulation results pretty well, although they're not perfect. This means that the approximation-based Eq. (5.3) could be used in practice. On the other hand, we observe some intriguing phenomena.

—In the large real or synthetic power-law graphs (Figures 4(a)–4(d)), node degree k has a nonlinear factor on the nodes' infection rate $i^*(k)$ whereas, for the small one (Figure 4(e)), node degree has a linear factor on $i^*(k)$. How can we interpret this discrepancy? We observe that the nonlinearity appears to be proportional to the range of the node degrees. For example, the nonlinearity in Figure 4(d) appears to be in between, and so is its node degree interval. This phenomenon arguably can be explained as that a nonlinear function could seem linear in a sufficiently small local region (e.g., degree range $[1, 250]$ in Figures 4(a)–4(d)).

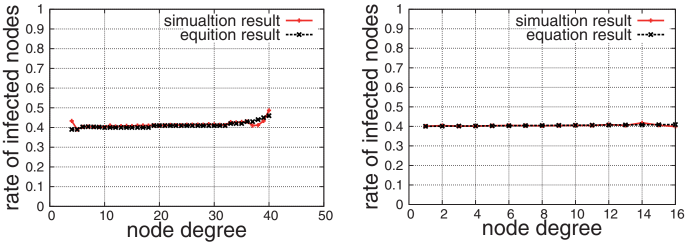
—We observe that for sufficient large degree k (e.g., when $k > 1,000$), $i^*(k)$ becomes almost flat. This can be explained as follows. We observe that the asymptotic limit $\lim_{k\to\infty} i^*(k) = 1/(1+\beta)$, which means that the variation of the slopes of the nonlinear



(a) Epinions dataset (large) (b) Enron dataset (large) (c) Oregon dataset (large)

(d) Power-law graph (large) (e) Power-law graph (small)

Fig. 4. On the accuracy of the approximation: Real and synthetic power-law graphs.



(a) Random graph (large) (b) Random graph (small)

Fig. 5. On the accuracy of the approximation: Synthetic random graphs.

curves vanishes rapidly and thus the curves look flat because (for example) the limit value is 0.635 for $\beta = 0.6$.

In the case of synthetic Erdos-Renyi random graphs (Figure 5), the numerical results and the simulation results agree to a large extent. We observe that node degree k has a linear factor on $i^*(k)$. Similar to the case of power-law graphs, this might be caused by the fact that the degrees are densely concentrated on a small interval, in which a nonlinear function could be approximated as a linear one.

In the case of synthetic regular graphs, we report that the numerical results and the simulation results match nicely. We did not plot the figures because there is only a single point in each figure, which does not offer any extra value.

In summary, we draw the following.

Insight 1. Node degree plays an essential role in governing node infection rate. Moreover, Eq. (5.3) is a reasonably good approximation of $i^*(k)$, which means that one could use Eq. (5.3) to estimate $i^*(k)$ for given parameters.

5.2. Condition under which the Mean Field Approach Is Accurate

In this section, we will provide further analyses of the push- and pull-based epidemic models using the well-known mean field approach, which is a statistical physics method introduced for the purpose of studying stochastic particle systems [Chandler 1987].

Roughly speaking, the mean field approach is to use the mean (expectation) to stand for stochastic interactions. Mathematically speaking, it can be seen as the first-order approximation of random variables. To be concrete, we here give a simple example. Suppose that a variable w is influenced by its random environment, which is described by a variable x, via a scheme h. That is, $w = h(x)$. By Taylor expansion, we have

$$w = h(x) = h(\langle x\rangle) + h'(\langle x\rangle)(x - \langle x\rangle) + o(|x - \langle x\rangle|),$$

where $\langle\cdot\rangle$ stands for the expectation (mean). By omitting the higher-order terms and conducting expectation on both sides, we have an approximation $\langle w\rangle = h(\langle x\rangle)$. This highlights the basic idea of the mean field approach: replacing a random field with its mean field.

The mean field approach also has become an important method for understanding complex dynamics. In general, this method is, however, not applicable in our setting because of the arbitrary network heterogeneity we accommodate. Therefore, it is natural to ask for conditions under which the method is applicable. In what follows we address this question.

We first derive a formula to describe $\bar{i}$, the global mean infection rate defined after the spreading becomes stable, namely,

$$\bar{i} = \langle i_v^*\rangle = \frac{1}{|V|}\sum_{v\in V} i_v^* = \frac{1}{|V|}\sum_{v\in V}\lim_{t\to\infty} i_v(t)$$

in Erdos-Renyi random graph with edge probability p. Recall the equation for the fixed point of master equation (3.2)

$$\beta i_v^* = \left[1-(1-\alpha)\prod_{(u,v)\in E}(1-\gamma i_u^*)\right](1-i_v^*).$$

So, we have

$$\beta\bar{i} = \left\langle\left[1-(1-\alpha)\prod_{(u,v)\in E}(1-\gamma i_u^*)\right](1-i_v^*)\right\rangle.$$

We use the following approximation to derive the right-hand side term.

$$\left\langle\left[1-(1-\alpha)\prod_{(u,v)\in E}(1-\gamma i_u^*)\right](1-i_v^*)\right\rangle \approx \left[1-(1-\alpha)\left\langle\prod_{(u,v)\in E}(1-\gamma i_u^*)\right\rangle\right](1-\bar{i})$$

In Erdos-Renyi random graphs with edge probability p, every pair of nodes are linked with probability p. So, we can approximate the following term as

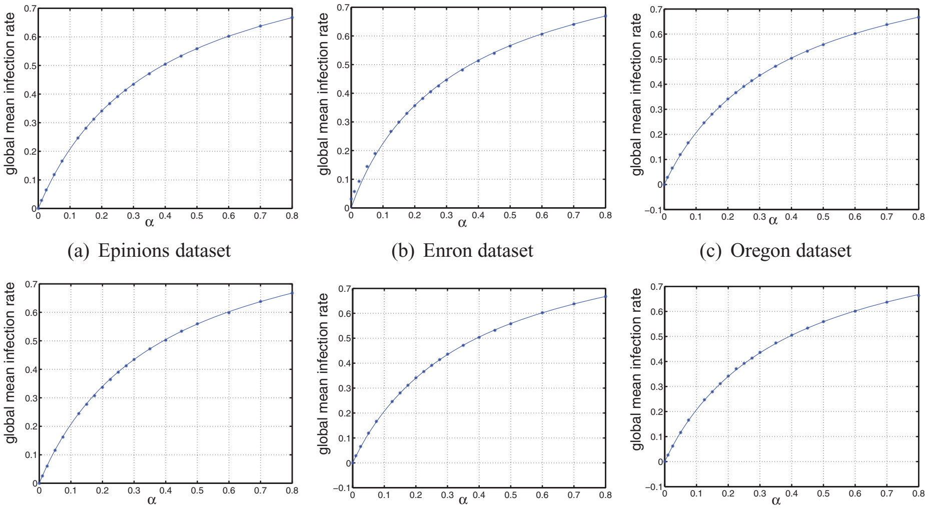
$$\left\langle\prod_{(u,v)\in E}(1-\gamma i_u^*)\right\rangle \approx (1-\gamma p\bar{i})^n.$$

Let $\langle k\rangle$ represent the average degree. When $\gamma\langle k\rangle$ is sufficiently small, we have

$$\log[(1-\gamma p\bar{i})^n] = n\log[1-\gamma p\bar{i}] \approx -np\gamma\bar{i} = -\gamma\langle k\rangle\bar{i}.$$

Thus, we have

$$\left\langle\prod_{(u,v)\in E}(1-\gamma i_u^*)\right\rangle \approx \exp(-\gamma\langle k\rangle\bar{i}).$$



(a) Epinions dataset (b) Enron dataset (c) Oregon dataset

(d) Power-law graph (small) (e) Random graph (small) (f) Regular graph (small)

Fig. 6. Mean-field approximation via Eq. (5.4) (the curve) vs. simulation (the dots): $\beta = 0.4$ and $\gamma = 0.004$. Observe that $\bar{i}$ increases with respect to α in a concave fashion.

So, we have

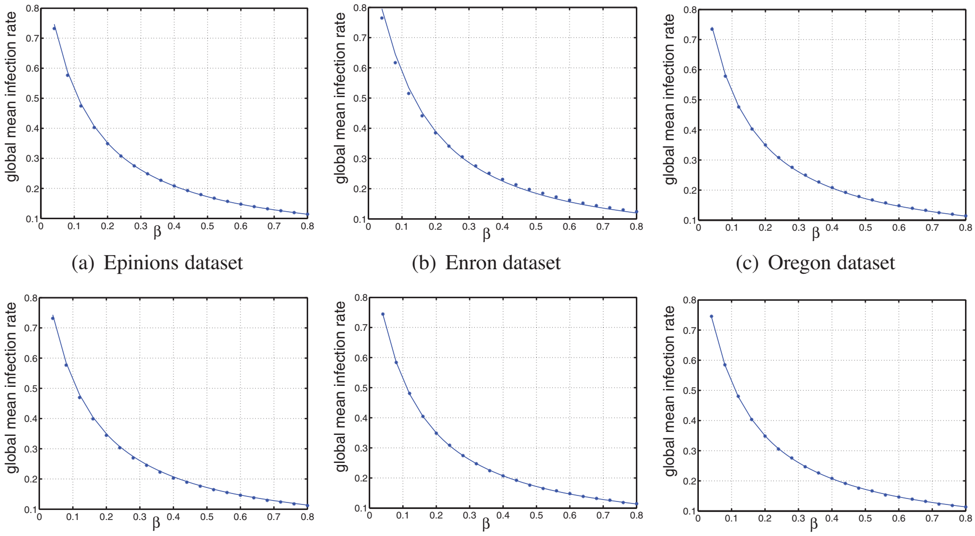
$$\beta\bar{i} \approx [1 - (1-\alpha)\exp(-\langle k\rangle\gamma\bar{i})](1-\bar{i}). \tag{5.4}$$

Its solution is $\bar{i}(\alpha, \beta, \gamma, \langle k\rangle)$. Although we are unable to derive its analytic expression, we can calculate the solution numerically.

The previous discussion was made for Erdos-Renyi random graphs with edge probability p with small $\gamma\langle k\rangle$. Can it be applied to other network topologies, perhaps under the condition $\gamma\langle k\rangle$ is small? To answer this question, we conduct simulation using some of the aforementioned datasets, which are selected with a certain balance on network size and topology (three large real datasets, three small synthetic datasets). Simulation results are based on the average of 50 runs.

Figure 6 compares the $\bar{i}$ derived from the simulation results after the spreading becomes stable (the dots) and the numerical solution $\bar{i}$ of Eq. (5.4) with fixed $\beta = 0.4$ and $\gamma = 0.004$. In the case of the small synthetic Erdos-Renyi random graph, for which Eq. (5.4) was derived, we observe that the simulation result and the numerical result match perfectly. Moreover, for the other network graphs, they also match nicely. In all cases, the global mean infection rate $\bar{i}$ increases with respect to α in a concave fashion. This means that in order to significantly lower the global mean infection rate, α must be decreased substantially (e.g., from $\alpha = 0.6$ down to $\alpha = 0.2$). In other words, security against pull-based infection such as “drive-by download” attacks must be substantially enhanced; otherwise, the spreading will remain at a high level.

Figure 7 compares the $\bar{i}$ derived from the simulation results after the spreading becomes stable (the dots) and the numerical solution $\bar{i}$ to Eq. (5.4) with fixed $\alpha = 0.1$ and $\gamma = 0.004$. In the case of Erdos-Renyi random graph, the simulation result and the numerical result match perfectly. For other network graphs, they also match well. In all cases, the global mean infection rate $\bar{i}$ decreases with respect to β in a convex fashion. This means that in order to lower the global mean infection rate significantly, β must be increased substantially (e.g., from $\beta = 0.1$ to $\beta = 0.5$). In other words, reactive security mechanisms against push- and pull-based infections must be substantially enhanced; otherwise, the spreading will remain at a relatively high level.



(a) Epinions dataset (b) Enron dataset (c) Oregon dataset

(d) Power-law graph (small) (e) Random graph (small) (f) Regular graph (small)

Fig. 7. Mean-field approximation via Eq. (5.4) (the curve) vs. simulation (the dots): $\alpha = 0.1$ and $\gamma = 0.004$. Observe that $\bar{i}$ decreases with respect to β in a convex fashion.

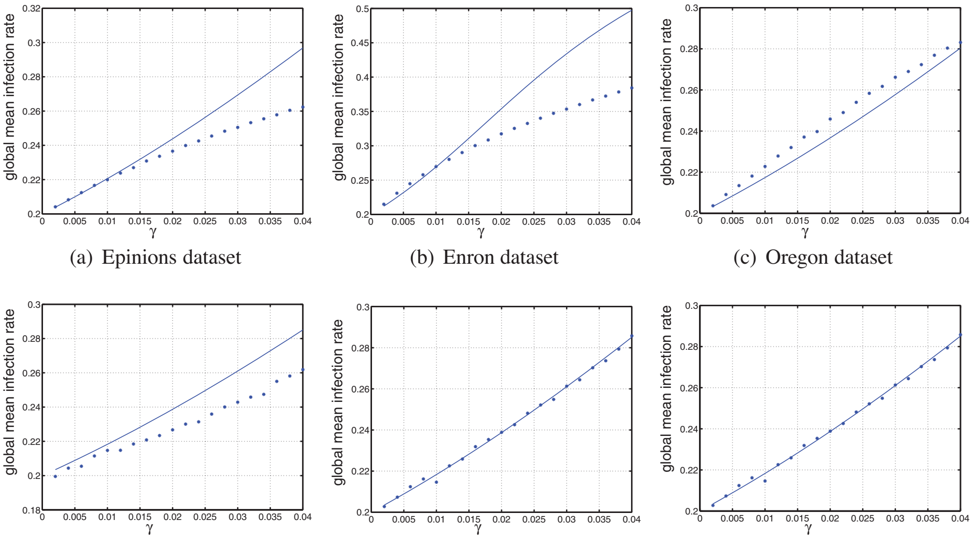
Figure 8 compares the $\bar{i}$ derived from the simulation results after the spreading becomes stable (the dots) and the numerical solution $\bar{i}$ to Eq. (5.4) with fixed $\alpha = 0.1$ and $\beta = 0.4$. In the case of random graph, the simulation result agrees with the numerical result. This also applies to the case of regular graph topology. In both cases, the global mean infection rate $\bar{i}$ increases with respect to γ in a somewhat linear fashion. However, for power-law topologies, the simulation result matches the numerical solution *only when* γ (therefore, $\gamma\langle k\rangle$) is small; this confirms the condition under which the mean field approach is applicable to arbitrary networks. In any case, we observe that any enhancement in security mechanisms against spreading (e.g., blocking suspicious or malicious sessions) will always have an almost linear impact on the mean infection rate. This is quite different from the impact of α and β because it states that network-based defense is always effective.

The preceding discussions lead us to draw the following.

Insight 2. The mean field approach is useful in the setting of arbitrarily heterogeneous networks when $\gamma\langle k\rangle$ is small. Under this circumstance, we found that the impacts of the parameters α, β, and γ are very different (as described before).

5.3. Peeking into Global Infection via Localized Monitoring

Estimating $\bar{i}$ via localized monitoring with unknown parameters. In the preceding we showed that when $\gamma\langle k\rangle$ is small, the solution to Eq. (5.4) is a good approximation of the global mean infection rate $\bar{i}$, Still, it requires to know the values of parameters α, β, and γ. This naturally leads to another question: What can we say or (how) can we obtain/approximate $\bar{i}$ even if the parameters are not known? In theory, we can derive $\bar{i}$ by monitoring the entire network so as to count the number of infected nodes at every time t. This is clearly infeasible in practice. It then becomes natural to ask: How can we sample the network so as to approximate $\bar{i}$ with a reasonable precision? Because the states of the nodes are *not* independent of each other, rather they are correlated in a certain delicate fashion as governed by the master equation (3.2), a reasonable



(a) Epinions dataset (b) Enron dataset (c) Oregon dataset

(d) Power-law graph (small) (e) Random graph (small) (f) Regular graph (small)

Fig. 8. Mean-field approximation via Eq. (5.4) (the curve) vs. simulation results (the dots): $\alpha = 0.1$ and $\beta = 0.4$. Observe that $\bar{i}$ increases with respect to γ in a somewhat linear fashion.

sampling strategy must take into consideration the evolution of the dynamics. In what follows we present such a result. To our surprise, the sample size can be constant and independent of the size of the network in question.

Recall that i_v^* is the infection rate of node v after the spreading becomes stable and $i^*(k)$ is the mean infection rate of the nodes with degree k. Then, we have

$$i^*(k) = \frac{1}{|\{v : \deg(v) = k\}|} \sum_{\deg(v)=k} i_v^* = \langle i_v^* | \deg(v) = k \rangle_v,$$

where $\langle \xi(u) | \Gamma \rangle_u$ denotes the conditional expectation of random variable $\xi(u)$ over $u \in V$ under condition Γ (the subscript u can be neglected if it is clear from the context). Recall Eq. (5.2), namely the approximation

$$\beta i^*(k) \approx [1 - (1-\alpha)(1 - \gamma i^*(k))^k](1 - i^*(k)).$$

Thus, the global mean infection rate equals to the mean of $i^*(k)$ over the degree distribution. We have

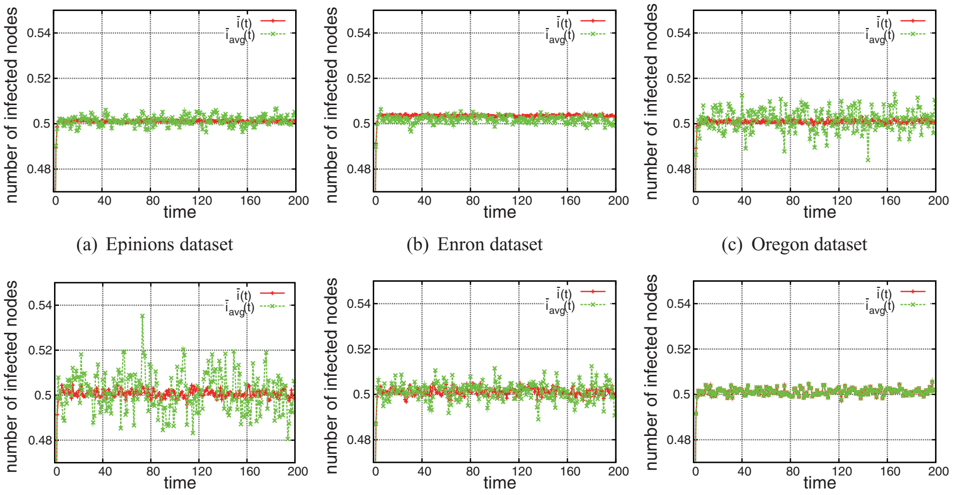
$$\bar{i} = \langle i_v^* \rangle_v = \langle i^*(k) \rangle_k,$$

due to the property of conditional expectation that $\langle \xi \rangle = \langle \langle \xi | \eta \rangle \rangle$ for two arbitrary random variables ξ and η. With linear approximation, we have

$$i^*(k) = i^*(\langle k \rangle) + \frac{di^*(k)}{dk}(k - \langle k \rangle) + o(k - \langle k \rangle),$$

which leads to

$$\bar{i} = \langle i^*(k) \rangle_k = i^*(\langle k \rangle) + \left\langle \frac{di^*(k)}{dk}(k - \langle k \rangle) \right\rangle_k + \big\langle o(k - \langle k \rangle) \big\rangle_k .$$



(a) Epinions dataset (b) Enron dataset (c) Oregon dataset

(d) Power-law graph (small) (e) Random graph (small) (f) Regular graph (small)

Fig. 9. $\bar{i}(t)$ vs. $\bar{i}_{\text{avg}}(t)$.

Neglecting the higher-order terms, assuming

$$\left\langle \frac{di^*(k)}{dk}(k - \langle k \rangle) \right\rangle_k \approx \left\langle \frac{di^*(k)}{dk} \right\rangle_k \langle (k - \langle k \rangle) \rangle_k,$$

and because $\langle k - \langle k \rangle \rangle_k = 0$, we have the following approximation

$$\bar{i} \approx i^*(\langle k \rangle).$$

This leads us to draw the following.

Insight 3. We can measure (or approximate) $\bar{i}$, the global mean infection rate in a network, through the mean infection rate of the nodes with the average node degree of a network.

To confirm the previous insight, we use simulation to compare the global mean infection rate $\bar{i}$ and the mean infection rate of the average-degree nodes, denoted by $\bar{i}_{\text{avg}}$. We plot the results in Figure 9. The parameters used for simulation are $\alpha = 0.4$, $\beta = 0.4$, and $\gamma = 0.001$; in each of these cases, the spreading is stable.

Figure 9 does show that they are reasonably close to each other at the resolution of 10^{-2}. The preceding insight is useful not only because the number of nodes with the average degree is much smaller than the total number of nodes, but also because we do not need to know the parameters α, β, and γ. In other words, it states that monitoring the nodes with the average degree is sufficient to measure or approximate the global state of the network; this explains why we call it *localized monitoring*.

Estimating $\bar{i}$ via localized monitoring with further reduced sample size. While the previous insight is already very useful, we ask a further deeper question: What if the number of nodes with the average degree is still large (e.g., thousands)? For example, in the case of the Epinions dataset, there are 1,070 nodes (that is, 1.41%) whose degrees equal to the average degree; in the case of Enron dataset, there are 2,212 nodes (that is, 6.03%) whose degrees equal to the average degree. Monitoring and examining whether those nodes are compromised at every time t is still costly. Is it possible to monitor even smaller number of nodes (ideally, constant number of nodes independent of the size of the network)? In what follows we answer this question affirmatively.

For the purpose of further reducing the number of nodes to monitor, we need to find further approximation to $i^*(\langle k\rangle)$. From the equations of the equilibrium.

$$\frac{\beta i_v^*}{1-i_v^*} = 1-(1-\alpha)\prod_{(u,v)\in E}(1-\gamma i_u^*),\ \ v \in V, \tag{5.5}$$

we know that the infection rate of node v depends on those of its neighboring nodes. For the same reason, the infection rate of node u, which is a neighbor of node v, depends on the state of u's neighboring nodes. Thus, we can regard i_v^* as a function of v's second-order neighborhood, which consists of nodes r such that $r \neq v \wedge (r, u) \in E \wedge (u, v) \in E$. More precisely, we define the *second-order degree* of node v, denoted by $\mathsf{2deg}(v)$, as

$$\mathsf{2deg}(v) = \sum_{u:(u,v)\in E} \deg(u) - |\{w : (w, v) \in E \wedge (v, w) \in E\}|.$$

This means that when two different neighbors of node v have a common neighbor r besides v itself, r's contribution to the infection rate of v should be counted twice. This leads us to consider the first- and second-order neighborhood of the nodes with the average degree $\langle k\rangle$. With the mean field approach, we have the following approximation.

$$\frac{\beta i_v^*}{1-i_v^*} \approx 1-(1-\alpha)(1-\gamma\langle i_u^*\rangle)^{\langle k\rangle}$$

and

$$\begin{aligned}\frac{\beta\langle i_u^*|(u,v)\in E\rangle}{1-\langle i_u^*|(u,v)\in E\rangle} &\approx \left\langle \frac{\beta i_u^*}{1-i_u^*}\middle|(u,v)\in E\right\rangle \\ &\approx 1-(1-\alpha)(1-\gamma\langle i_r^*\rangle_{r:r\neq v\wedge(r,u)\in E\wedge(u,v)\in E})^{\langle \deg(u)|(u,v)\in E\rangle}\end{aligned}$$

As a result, we can further approximately regard i_v^* with $\deg(v) = \langle k\rangle$ as a random variable over

$$\mathsf{2deg}(v) = [\langle \deg(u)|(u,v)\in E\rangle_u - 1]\deg(v).$$

Assuming $\langle i_r^*|(r,u)\in E,\ (u,v)\in E, \deg(v) = \langle k\rangle\rangle_v = i_v^*$ and letting $\tilde{i}_v = \langle i_u^*|(u,v)\in E\rangle$, we have

$$\frac{\beta i_v^*}{1-i_v^*} = 1-(1-\alpha)(1-\gamma\tilde{i}_v)^{\langle k\rangle} \quad \text{and} \quad \frac{\beta\tilde{i}_v}{1-\tilde{i}_v} = 1-(1-\alpha)(1-\gamma i_v^*)^{k'_v},$$

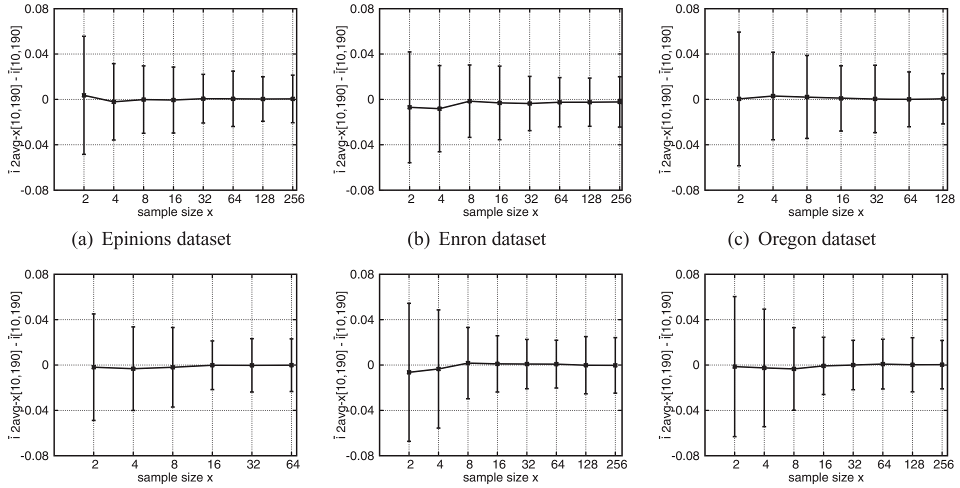
where

$$k'_v = \frac{\mathsf{2deg}(v)}{k}.$$

Given the mean degree $\langle k\rangle$, we can regard i_v^* as a function of random variable k'_v, that is, $i_v^* = f(k')$ for all v with $\deg(v) = k$ and for some function $f(\cdot)$. By a discussion similar to before, we have an approximation $\bar{i} \approx f(\langle k'\rangle) = f(\langle \mathsf{2deg}(v)\rangle_{v:\deg(v)=k}/\langle \deg(v)\rangle)$. This means that we can select the nodes whose degrees equal to $\langle k\rangle$ and second-order degrees equal to $\langle \mathsf{2deg}(v)|\deg(v) = \langle k\rangle\rangle$. This leads us to draw the following insight, which allows to further reduce the sample size.

Insight 4. We can approximate $\bar{i}$, the global mean infection rate, by only monitoring the nodes whose degrees equal to the average degree $\langle k\rangle$ and second-order degrees equal to $\langle \mathsf{2deg}(v)|\deg(v) = \langle k\rangle\rangle$.

The preceding insight statistically states that if there are no or few nodes whose degrees and second-order degrees satisfy the requirements, then we can select the



(a) Epinions dataset (b) Enron dataset (c) Oregon dataset

(d) Power-law graph (small) (e) Random graph (small) (f) Regular graph (small)

Fig. 10. $\bar{i}_{\text{2avg-x}}[10, 190] - \bar{i}[10, 190]$.

nodes whose degrees and second-order degrees are close to the respective average degrees. Because the insights do not specify how large the reduced sample should be, in what follows we use simulation to demonstrate how large the samples need be.

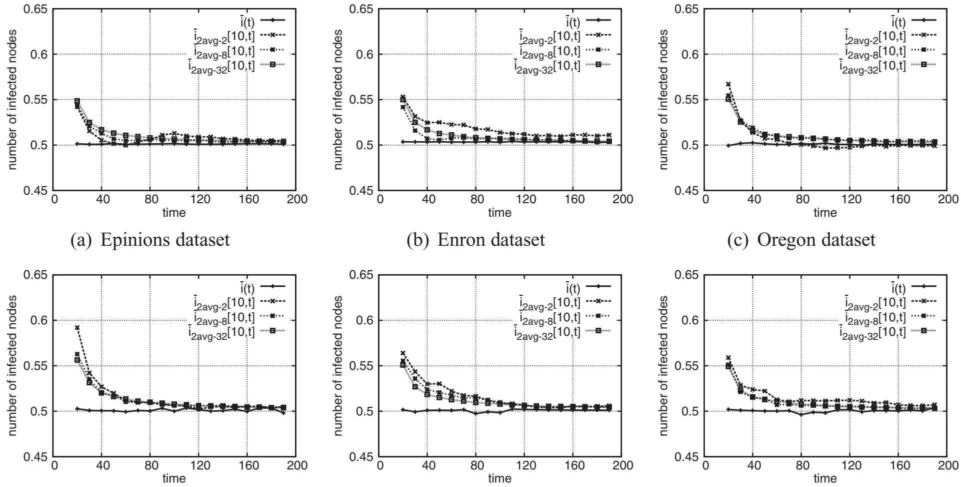
Let 2avg-x denote the set of x nodes whose degrees equal to the average degree and second-order degrees are the closest to $\langle \text{2deg}(v) | \deg(v) = \langle k \rangle \rangle$. We note that there are two ways to approximate $\bar{i}$ via $\bar{i}_{\text{2avg-x}}$.

—We approximate $\bar{i}[t_0, t_1] = \frac{1}{t_1 - t_0 + 1} \sum_{t \in [t_0, t_1]} \bar{i}(t)$ as $\bar{i}_{\text{2avg-x}}[t_0, t_1] = \frac{1}{t_1 - t_0 + 1} \sum_{t \in [t_0, t_1]} \sum_{w \in \text{2avg-x}} i_w(t)$. This approximation is meaningful because the former captures the global mean infection rate averaged over time interval $[t_0, t_1]$ and the latter captures the mean infection rate of the x sampled nodes over the same period of time. Note that $t_0 > 0$ is used because we deal with the situation after the spreading becomes stable; our simulation result indicates that t_0 can be no greater than 10.

—We approximate $\bar{i}(t)$ via $\bar{i}_{\text{2avg-x}}[t_0, t] = \frac{1}{t - t_0 + 1} \sum_{t \in [t_0, t]} \sum_{w \in \text{2avg-x}} i_w(t)$. This approximation is useful and interesting because the latter captures the "history" of the small samples, which can actually approximate a certain index of the whole network.

Corresponding to the first type of approximation mentioned earlier, Figure 10 plots $\bar{i}_{\text{2avg-x}}[t_0, t_1] - \bar{i}[t_0, t_1]$ over time interval $[t_0, t_1] = [10, 190]$ (with step-length 10) based on our simulation study, where $x \in \{2, 4, 8, 16, 32, 64, 128, 256\}$ nodes are selected as indicated before. We observe that in all the plotted network graphs, 16 nodes are enough to form a sufficiently accurate sample. What is surprising is that it seems a constant size of samples is sufficient, regardless of the size of the graph. It is an exciting future work to analytically explain this phenomenon.

Corresponding to the second type of approximation mentioned earlier, Figure 11 plots $\bar{i}_{\text{2avg-x}}[t_0, t]$ versus. $\bar{i}(t)$, where $t_0 = 10$, $t = 20, 30, \ldots, 200$, and $x \in \{2, 8, 32\}$ nodes are selected as indicated previously. Note that the accuracy shown in Figure 11 is not as good as the accuracy shown in Figure 10. This is because Figure 10 corresponds to the average over time interval $[10, 190]$, where the comparison is between $\bar{i}[10, 190]$ and $\bar{i}_{\text{2avg-x}}[10, 190]$, whereas Figure 11 corresponds to $\bar{i}(t)$ and $\bar{i}_{\text{2avg-x}}[10, t]$ for



(a) Epinions dataset (b) Enron dataset (c) Oregon dataset

(d) Power-law graph (small) (e) Random graph (small) (f) Regular graph (small)

Fig. 11. $\bar{i}_{\text{2avg-x}}[10, t]$ vs. $\bar{i}(t)$.

$t = 10, 20, \ldots, 190$. Nevertheless, as shown in Figure 11, for moderate t (e.g., $t = 120$), the prediction of the future is with reasonable accuracy. This leads to the following.

Insight 5. The accumulative observations obtained from monitoring certain constant number of nodes (selected as mentioned before, for example, monitoring 32 nodes as shown in Figure 11) allow to estimate the current global mean infection rate $\bar{i}(t)$ with a pretty good accuracy.

6. CONCLUSION AND FUTURE WORK

We rigorously investigated a push- and pull-based epidemic spreading model in arbitrary networks. We presented sufficient conditions or epidemic thresholds under which the spreading will become stable (but not dying out). The conditions supersede the relevant ones offered in the literature. Our investigation leads to further insights regarding the characterization of the role of node degree in governing node infection rate, the characterization of a condition under which the mean field approach is applicable in arbitrary networks, and a sampling strategy for selecting a small number of nodes to monitor so as to estimate the global mean infection rate without knowing the values of the parameters.

There are many exciting research problems from both theoretic and practical points of view. In addition to those mentioned in the text, here we highlight the following. First, how can we precisely pin down the equilibrium states in the general case of $\alpha > 0$? Second, can we have tighter bounds especially when the spreading is not stable? Third, can we statistically prove that the sample size can be constant and thus independent of the size of the network (as indicated by our empirical study)? Fourth, can we design sampling algorithms to sample the nodes *without* knowing the network topology (our characterization demonstrates their existence but requires the knowledge of the global network)?

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments that helped us improve the article.

REFERENCES

ANDERSON, R. AND MAY, R. 1991. *Infectious Diseases of Humans*. Oxford University Press.

BAILEY, N. 1975. *The Mathematical Theory of Infectious Diseases and Its Applications*, 2^{nd} Ed. Griffin, London.

BARABASI, A. AND ALBERT, R. 1999. Emergence of scaling in random networks. *Science 286,* 509–512.

BARRAT, A., BARTHLEMY, M., AND VESPIGNANI, A. 2008. *Dynamical Processes on Complex Networks*. Cambridge University Press.

CALLAWAY, D. S., NEWMAN, M. E. J., STROGATZ, S. H., AND WATTS, D. J. 2000. Network robustness and fragility: Percolation on random graphs. *Phys. Rev. Lett. 85*, 25, 5468–5471.

CHAKRABARTI, D., WANG, Y., WANG, C., LESKOVEC, J., AND FALOUTSOS, C. 2008. Epidemic thresholds in real networks. *ACM Trans. Inf. Syst. Secur. 10*, 4, 1–26.

CHANDLER, D. 1987. *Introduction to Modern Statistical Mechanics*. Oxford University Press.

CHEN, Z. AND JI, C. 2005. A self-learning worm using importance scanning. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM'05).* 22–29.

DEMERS, A., GREENE, D., HAUSER, C., IRISH, W., LARSON, J., SHENKER, S., STURGIS, H., SWINEHART, D., AND TERRY, D. 1987. Epidemic algorithms for replicated database maintenance. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'87).* 1–12.

FALOUTSOS, M., FALOUTSOS, P., AND FALOUTSOS, C. 1999. On power-law relationships of the internet topology. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures and Protocols for Computer Communication (SIGCOMM'99).* 251–262.

GANESH, A., MASSOULIE, L., AND TOWSLEY, D. 2005. The effect of network topology on the spread of epidemics. In *Proceedings of the IEEE International Conference on Computer Communications (InfoCom'05).*

HETHCOTE, H. 2000. The mathematics of infectious diseases. *SIAM Rev. 42*, 4, 599–653.

HORN, R. AND JOHNSON, C. 1985. *Matrix Analysis*. Cambridge University Press.

KARP, R., SCHINDELHAUER, C., SHENKER, S., AND VÖCKING, B. 2000. Randomized rumor spreading. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'00).* 565–574.

KEMPE, D. AND KLEINBERG, J. 2002. Protocols and impossibility results for gossip-based communication mechanisms. In *Proceedings of the 43^{rd} Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*. 471–480.

KEMPE, D., KLEINBERG, J., AND DEMERS, A. 2001. Spatial gossip and resource location protocols. In *Proceedings of the Symposium on the Theory of Computing (STOC'01).* 163–172.

KEPHART, J. AND WHITE, S. 1991. Directed-Graph epidemiological models of computer viruses. In *Proceedings of the IEEE Symposium on Security and Privacy.* 343–361.

KEPHART, J. AND WHITE, S. 1993. Measuring and modeling computer virus prevalence. In *Proceedings of the IEEE Symposium on Security and Privacy*. 2–15.

KERMACK, W. AND MCKENDRICK, A. 1927. A contribution to the mathematical theory of epidemics. *Proc. Roy. Soc. Lond. A115*, 700–721.

LI, X., PARKER, P., AND XU, S. 2007. Towards quantifying the (in)security of networked systems. In *Proceedings of the 21^{st} IEEE International Conference on Advanced Information Networking and Applications (AINA'07).* 420–427.

MCKENDRICK, A. 1926. Applications of mathematics to medical problems. *Proc. Edin. Math. Soc. 14*, 98–130.

MEDINA, A., LAKHINA, A., MATTA, I., AND BYERS, J. 2001. Brite: An approach to universal topology generation. In *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'01).* 346–356.

MOLLOY, M. AND REED, B. 1995. A critical point for random graphs with a given degree sequence. *Rand. Struct. Algor. 6,* 161–179.

MOLLOY, M. AND REED, B. 1998. The size of the giant component of a random graph with a given degree sequence. *Comb. Probab. Comput. 7*, 295–305.

MORENO, Y., PASTOR-SATORRAS, R., AND VESPIGNANI, A. 2002. Epidemic outbreaks in complex heterogeneous networks. *Euro. Phys. J. B26*, 521–529.

NEWMAN, M. 2003. The structure and function of complex networks. *SIAM Rev. 45,* 167.

NEWMAN, M. 2007. Component sizes in networks with arbitrary degree distributions. *Phys. Rev. E76*, 4, 045101.

NEWMAN, M. 2010. *Networks: An Introduction*. Oxford University Press.

NEWMAN, M. E. J., STROGATZ, S. H., AND WATTS, D. J. 2001. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E64*, 2, 026118.

PASTOR-SATORRAS, R. AND VESPIGNANI, A. 2001. Epidemic dynamics and endemic states in complex networks. *Phys. Rev. E63*, 066117.

PASTOR-SATORRAS, R. AND VESPIGNANI, A. 2002. Epidemic dynamics in finite size scale-free networks. *Phys. Rev. E65*, 035108.

PROVOS, N., MCNAMEE, D., MAVROMMATIS, P., WANG, K., AND MODADUGU, N. 2007. The ghost in the browser analysis of web-based malware. In *Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (HotBots'07).*

SHAH, D. 2009. Gossip algorithms. *Found. Trends Netw. 3,* 1, 1–125.

WANG, Y., CHAKRABARTI, D., WANG, C., AND FALOUTSOS, C. 2003. Epidemic spreading in real networks: An eigenvalue viewpoint. In *Proceedings of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS'03)*. 25–34.

WILF, H. 1994. *Generating Functionology.* Academic Press.

WILLINGER, W., ALDERSON, D., AND DOYLE, J. 2009. Mathematics and the internet: A source of enormous confusion and great potential. *Not. Amer. Math. Soc. 56*, 5, 286–299.

YOSHIDA, K. 1971. *Functional Analysis and Its Applications*. Springer.

ZOU, C., GAO, L., GONG, W., AND TOWSLEY, D. 2003. Monitoring and early warning for internet worms. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'03),* V. Atluri, Ed. ACM Press, New York, 190–199.

ZOU, C., GONG, W., AND TOWSLEY, D. 2002. Code red worm propagation modeling and analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02).* 138–147.

Received October 2010; revised April 2011; accepted June 2011

Adaptive Epidemic Dynamics in Networks: Thresholds and Control

Shouhuai Xu
University of Texas at San Antonio
Wenlian Lu
Fudan University
and
Li Xu and Zhenxin Zhan
University of Texas at San Antonio

Theoretical modeling of computer virus/worm epidemic dynamics is an important problem that has attracted many studies. However, most existing models are adapted from biological epidemic ones. Although biological epidemic models can certainly be adapted to capture some computer virus spreading scenarios (especially when the so-called homogeneity assumption holds), the problem of computer virus spreading is not well understood because it has many important perspectives that are not necessarily accommodated in the biological epidemic models. In this paper we initiate the study of such a perspective, namely that of *adaptive* defense against epidemic spreading in arbitrary networks. More specifically, we investigate a non-homogeneous Susceptible-Infectious-Susceptible (SIS) model where the model parameters may vary with respect to time. In particular, we focus on two scenarios we call *semi-adaptive* defense and *fully-adaptive* defense, which accommodate implicit and explicit dependency relationships between the model parameters, respectively. In the semi-adaptive defense scenario, the model's input parameters are given; the defense is semi-adaptive because the adjustment is implicitly dependent upon the outcome of virus spreading. For this scenario, we present a set of sufficient conditions (some are more general or succinct than others) under which the virus spreading will die out; such sufficient conditions are also known as *epidemic thresholds* in the literature. In the fully-adaptive defense scenario, some input parameters are not known (i.e., the aforementioned sufficient conditions are not applicable) but the defender can observe the outcome of virus spreading. For this scenario, we present adaptive control strategies under which the virus spreading will die out or will be contained to a desired level.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: computer malware, virus epidemics, epidemic dynamics, epidemic threshold, complex network, graph

Author's address: Shouhuai Xu, Li Xu, and Zhenxin Zhan are with the Department of Computer Science, University of Texas at San Antonio. Corresponding author: Shouhuai Xu (shxu@cs.utsa.edu).
Wenlian Lu is with the Center for Computational Systems Biology and the School of Mathematical Sciences, Fudan University (wenlian@fudan.edu.cn).

1. INTRODUCTION

Theoretically modeling the spreading dynamics of computer virus (or malware such as worm and bot) is important for deepening our understanding and for designing effective, if not optimal, defenses. We observe, however, that the utility of theoretical modeling in this context is not well understood yet because existing models are often adapted from biological epidemic ones. As a consequence, many existing models of computer virus spreading dynamics made the so-called *homogeneity* assumption, which roughly says that the nodes are equally powerful in infecting others. Realizing the limitation of the assumption, there have been investigations that aim to weaken the assumption by considering heterogeneous network topology (where different nodes may have different infection capabilities because they have different degrees). Along this line of study, the present paper moves a step further by exploring models that accommodate realistic scenarios where the model parameters may change over time (i.e., the parameters are some functions of time), which captures the fact that both attack and defense are dynamically evolving or under dynamical adjustment and reflects the persistence of virus spreading. This allows us to investigate an important and novel perspective of virus spreading-defense dynamics, namely that of *adaptive* defense against computer virus spreading.

1.1 Our Contributions

We investigate a non-homogeneous Susceptible-Infectious-Susceptible (SIS) model in arbitrary networks (i.e., there is no restriction on the topology of the spreading networks and the nodes may have different defense or cure capabilities). The model can accommodate both *semi-adaptive* defense and *fully-adaptive* defense. In the semi-adaptive defense scenario, the input parameters in the model are known and can vary with respect to time (e.g., according to some deterministic functions of time or according to some stochastic process, but we do not impose any practical restrictions on the types of functions). For this scenario, we present a set of sufficient conditions, from general to specific (but more succinct), under which the virus spreading will die out. We note that such sufficient conditions are also known as *epidemic thresholds* in the literature.

In the fully-adaptive defense scenario, some input parameters are not known and thus the aforementioned sufficient conditions are not applicable. Nevertheless, the defender might be able to observe the outcome of virus spreading (i.e., which nodes are infected at a point in time). For this scenario, we present adaptive control strategies under which the virus spreading will die out or will be contained to a desired level (which is important when, for example, the price to kill the virus spreading may be too high).

Because of the above, our model supersedes previous homogeneous and non-homogeneous models that offered relevant analytical insights; the concrete connection will be made when the need arises. Our analytical results are confirmed via simulation, from which we draw additional observations that serve as hints for future modeling studies. We discuss the practical implications of our model and the derived insights as well.

Finally, we note that the present paper is meant to explore theoretical characterizations of spreading-defense dynamics while assuming certain parameters can be observed or measured (e.g., based on extensive data and possibly expert knowledge). This may not be feasible some times. Regardless, we believe that such studies are important on their own and represent a necessary step towards the ultimate characterization of virus spreading-defense dynamics (which in turn helps design more effective or even optimal defenses).

1.2 Related Work

To the best of our knowledge, there are no existing studies on modeling adaptive spreading-defense dynamics in arbitrary networks. The work that is most closely related to ours was due to Chakrabarti et al. [Chakrabarti et al. 2008], who considered computer virus spreading in arbitrary networks — a scenario also investigated in [Wang et al. 2003; Ganesh et al. 2005]. The most important contribution of these studies is the identification of a sufficient condition (i.e., epidemic threshold) under which the virus spreading will die out; we will discuss the relationship between their result and ours when the need arises. Earlier studies either made the homogeneity assumption (e.g., [Kephart and White 1991; 1993]) as in biological epidemic models (see, for example, [McKendrick 1926; Kermack and McKendrick 1927; Bailey 1975; Anderson and May 1991; Hethcote 2000]), or considered specific non-homogeneous networks [Chakrabarti et al. 2008].

We should mention prior work that is conceptually or spiritually relevant. The concept of "adaptable robust computer systems" was investigated by Bhargava et al. [Bhargava et al. 1986], which however has a very different meaning and is for very different purposes. Also for a different purpose, Zou et al. [Zou et al. 2005] explored the concept of "adaptive defense" based on cost optimization, where cost was introduced by false positives and false negatives. In particular, they considered optimal adaptive defense against worm infection, but is from the perspective of decision whether or not to block/allow some specific host traffic. As such, it may be possible to combine their studies and ours because we do not consider cost.

Outline: In Section 2 we present our model as well as the analytical insights. We report our simulation study in Section 3. We conclude the paper in Section 4 with open problems.

2. ADAPTIVE EPIDEMIC DYNAMICS: MODEL AND ANALYSIS

2.1 The model

Primary parameters. Because we want to accommodate spreading in arbitrary networks, we assume that virus spreads over a series of finite, dynamical graphs $G(t) = (V, E(t))$, where V, $|V| = n$, is the set of nodes or vertices and $E(t)$ is the set of (possibly changing) edges or arcs at time $t \geq 0$ (i.e., the topology may change with respect to time). At any time t, an infected node u can directly infect node v if $(u, v) \in E(t)$. Denote by $A(t) = [a_{vu}(t)]$ the adjacency matrix of $G(t)$, where $a_{vv}(t) = 0$ for all $v \in V$, and $a_{vu}(t) = 1$ if and only if $(u, v) \in E(t)$. Note that this representation naturally accommodates both directed and undirected topologies, and thus our results equally apply to them.

A node $v \in V$ is susceptible if v is secure but vulnerable, and infected if v is successfully attacked (i.e., infected and infectious). At any time t, a node $v \in V$ is either susceptible or infected. Moreover, a susceptible node may become infected because of some infected node u where $(u, v) \in E(t)$, and an infected node may become susceptible because of cure. Since an infected node may become susceptible again, our model falls into the category of the so-called SIS models, but our model has the unique feature that values of the parameters can change with respect to time.

We consider two dependent variables: $s_v(t)$, the probability $v \in V$ is susceptible at time t; $i_v(t)$, the probability $v \in V$ is infected at time t. We consider a continuous-time model, which preserves the invariant $s_v(t) + i_v(t) = 1$. The model's input parameters are:

—$\beta_v(t)$: The probability an infected node v becomes susceptible at time t.

—$\gamma_{uv}(t)$: The probability an infected node u successfully infects a susceptible node v over edge $(u,v) \in E(t)$ at time t. For simplicity, we assume that $\gamma_{uv}(t) = \gamma(t)$ for all $(u,v) \in E(t)$.

For the sake of mathematical rigorousness, the $\beta_v(t)$'s and the $\gamma_{uv}(t)$'s, which are probabilities, should be "measurable" so as to ensure the existence of solutions to system (2.4) and be "bounded" so as to ensure the proof of Theorem 2 can get through. To avoid any unnecessary mathematical subtleties, we simply assume that these parameters are "boundedly measurable," which has no consequence in practice. Note that $A(t)$ is naturally bounded.

Other parameters and notations. Below is a summary of the major notations used in the paper; notations only occasionally used are explained when the need arises.

model input parameters:	
$A(t) = [a_{vu}(t)]$	the adjacency matrix of graph $G(t) = (V, E(t))$ where $\lvert V \rvert = n$, and $a_{vu}(t) = 1$ if and only if $(u,v) \in E(t)$. Moreover, $a_{vv}(t) = 0$ for all $v \in V$.
$\beta_v(t)$	the cure capability of node v at time t
$\gamma(t)$	the edge infection capability at time t
dependent variables:	
$s_v(t)$	the probability node $v \in V$ is susceptible at time t
$i_v(t)$	the probability node $v \in V$ is infected at time t
intermediate variables:	
$\delta_v(t)$	the probability susceptible $v \in V$ becomes infected at time t because of infected neighbors $\{u : (u,v) \in E(t)\}$
other notations:	
λ_1	the largest (in modulus) eigenvalue of adjacency matrix A
$\lVert \cdot \rVert$	the 1-norm of vector or matrix
$B(t) = \text{diag}[\beta_1(t), \cdots, \beta_n(t)]$	the cure probability diagonal matrix

The state transition diagram and master equation. Figure 1 depicts the state transition diagram of a node, where the probability $\delta_v(t)$ is given by

$$\delta_v(t) = 1 - \prod_{(u,v) \in E(t)} \left[1 - \gamma(t) \cdot i_u(t)\right]. \tag{2.1}$$

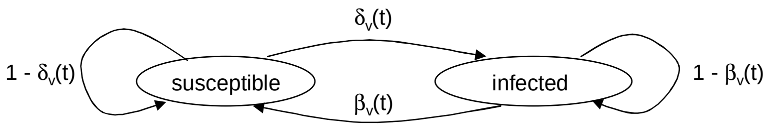


Fig. 1. State transition diagram of node $v \in V$ at time t

Note that in the derivation of Eq. (2.1), we assumed that the events that infected neighbors infect a node are independent. Note also that $s_v(t) + i_v(t) = 1$ for any t. Based on the state transition diagram we obtain the following master equation of dynamics:

$$\frac{di_v(t)}{dt} = \left[1 - \prod_{u \in V} [1 - \gamma(t) a_{vu}(t) i_u(t)]\right] [1 - i_v(t)] - \beta_v(t) i_v(t), \tag{2.2}$$

where $a_{vu}(t) = 1$ if and only if $(u, v) \in E(t)$.

2.2 Sufficient conditions for dying out in the scenario of semi-adaptive defense

In this subsection we present a set of sufficient conditions under which the virus spreading will die out. The sufficient conditions are applicable when the model's input parameters, namely the $\beta_v(t)$'s and $\gamma(t)$ are given. Moreover, it is possible that $\beta_v(t)$ relies on $\gamma(t)$; for example, the former is an (implicit) function of the latter. This explains why we call this scenario the semi-adaptive defense.

THEOREM 1. (a general sufficient condition under which virus spreading dies out) *Consider the following comparison (and linearization) system of Eq. (2.2):*

$$\frac{dx_v(t)}{dt} = \sum_{u=1}^{n} a_{vu}(t) \gamma(t) x_u(t) - \beta_v(t) x_v(t). \tag{2.3}$$

Let $x(t) = [x_1(t), \cdots, x_n(t)]^\top$, we obtain the following compact form of Eq. (2.3):

$$\frac{dx(t)}{dt} = \left[\gamma(t) A(t) - B(t)\right] x(t). \tag{2.4}$$

Denote by $U(t, t')$ the solution matrix of linear system (2.4), namely that each solution of linear system (2.4), $x(t)$, with initial condition $x(t_0) = x_0$, can be written as $x(t) = U(t, t_0) x_0$. Because the solution $x(t)$ is dependent upon the initial value x_0 but the solution matrix $U(t, t')$ is not, the corresponding maximum Lyapunov exponent (MLE) is determined by the solution matrix (rather than by the solution $x(t)$) and can be defined as [Oseledec 1968]:

$$\mu = \lim_{t \to \infty} \frac{1}{t} \ln \|U(t, 0)\|$$

where $\|\cdot\|$ is (for specificality) the 1-norm of matrix (because μ is independent of the choice of the norm). If $\mu < 0$, the virus spreading will die out regardless of the initial infection configuration[1]*; if $\mu > 0$ and system (2.4) is ergodic [Oseledec 1968], the virus spreading will not die out in some initial infection configurations (i.e., "the equilibrium of $i^* = 0$ is unstable" in mathematical terms).*

PROOF. Note that

$$1 - \prod_{u \in V} [1 - \gamma(t) a_{vu}(t) i_u(t)] \le \gamma(t) \sum_{u \in V} a_{vu}(t) i_u(t). \tag{2.5}$$

If $i_v(0) = x_v(0)$ for all $v \in V$, then the comparison system (2.3) satisfies that $i_v(t) \le x_v(t)$ holds for all $t \ge 0$ and $v \in V$. Since Eq. (2.3) is actually the linear system of the system (2.2) at $i_v = 0$ for all $v \in V$. Therefore, we can conclude that the stability of Eq. (2.2) is equivalent to that of Eq. (2.3). In other words, if all $x_v(t)$'s of system (2.3) converge to zero, then system (2.2) is stable regardless of the initial values; on the

[1] An initial infection configuration is specified by "which, and thus how many, nodes are initially infected".

other hand, if system (2.3) is unstable, then system (2.2) is also unstable. If system (2.4) is ergodic [Oseledec 1968], then the limit μ exists; otherwise, we can alternatively define $\mu = \overline{\lim}_{t\to\infty} \frac{1}{t} \ln \|U(t,0)\|$, where $\overline{\lim}_{t\to\infty} z(t)$ represents the upper bound of the limit of $z(t)$ as t goes to infinity, is also guaranteed to exist. In any case, by applying the definition of MLE, we obtain the theorem immediately. □

Discussion. The above sufficient condition $\mu < 0$ for the virus spreading to die out is actually close to being necessary, meaning that if $\mu > 0$, then the virus spreading will not die out in *most*, rather than just *some*, initial infection configurations. According to the Lyapunov exponent and smooth ergodic theory developed by [Pesin 1977] and many others, $\mu > 0$ means that the system (2.2) possesses an unstable manifold, which implies that the stable manifold, i.e., the set of points (i.e., the initial values) starting from which the system (2.2) converges to the origin, has dimension less than n. Therefore, the stable manifold has Lesbegue measure 0. That is, except a set with Lesbegue measure 0, the virus spreading never dies out with respect to any initial infection configuration.

The sufficient condition given in Theorem 1 is very general because in its derivation no "amplification" is used and dynamical topology $E(t)$ is accommodated. However, it requires to, among other things, solve a system of n linear equations of n variables (equivalently, diagonalizing a $n \times n$ matrix), which can be quite time-consuming for large n (the number of nodes). In what follows we give two succinct sufficient conditions, which can be easily connected to previous state-of-the-art results.

THEOREM 2. (a succinct sufficient condition) *Suppose $\beta_v(t) = \beta(t)$ for all $v \in V$ (i.e., all nodes have the same cure capability) and $E(t) = E$ for any time t, meaning that the topology does not change over time and $A = A(t)$ for any t. Consider the system*

$$\frac{dx(t)}{dt} = \Big[\gamma(t)A - \beta_v(t)I_n\Big]x(t), \tag{2.6}$$

where I_n is the $n \times n$ identity matrix. Let

$$\bar{\gamma} = \lim_{t\to\infty} \frac{1}{t}\int_{t_0}^{t+t_0} \gamma(\tau)d\tau, \quad \bar{\beta} = \lim_{t\to\infty} \frac{1}{t}\int_{t_0}^{t+t_0} \beta(\tau)d\tau,$$

and suppose the limits exist and are uniform with respect to t_0. Let λ_1 be the largest (in modulus) eigenvalue of A. If

$$\lambda_1 < \frac{\bar{\beta}}{\bar{\gamma}}, \tag{2.7}$$

then the virus spreading will die out regardless of the initial infection configuration; if

$$\lambda_1 > \frac{\bar{\beta}}{\bar{\gamma}}, \tag{2.8}$$

then the virus spreading will not die out in some initial infection configurations.

PROOF. Consider system (2.4). Let $A = S^{-1}JS$ be the Jordan canonical form with

$$J = \begin{bmatrix} J_1 & & & & \\ & J_2 & & & \\ & & J_3 & & \\ & & & \ddots & \\ & & & & J_K \end{bmatrix}, \; J_k = \begin{bmatrix} \lambda_k & 0 & 0 & \cdots & 0 \\ 1 & \lambda_k & 0 & \cdots & 0 \\ 0 & 1 & \lambda_k & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & \lambda_k \end{bmatrix},$$

where λ_k, $k = 1, \cdots, K$, are the distinct eigenvalues of A. Recall that λ_1 is the largest eigenvalue in modulus. From the Perron-Frobenius theorem [Berman and Shaked-Monderer 2003], λ_1 is a real number. Then, letting $y(t) = Sx(t)$, we have

$$\frac{dy(t)}{dt} = [\gamma(t)J - \beta(t)I_n]y(t).$$

Namely,

$$\frac{dy_v(t)}{dt} = [\gamma(t)\lambda_{k_v} - \beta(t)]y_v(t) + \xi_v(t),$$

where λ_{k_v} is the eigenvalue of A corresponding to the Jordan block J_{k_v} that contains column v, and $\xi_v(t) = 0$ if the v-th row of S is an eigenvector of A and $\xi_v(t) = \gamma(t)y_{v-1}(t)$ otherwise. First, consider $v = 1$ corresponding to the eigenvalues λ_1. We have

$$y_1(t) = y_1(0)\exp\left(\int_0^t [\gamma(\tau)\lambda_1 - \beta(\tau)]d\tau\right). \tag{2.9}$$

One can see that the Lyapunov exponent of system (2.9) is calculated as

$$\begin{aligned} \lim_{t\to\infty} \frac{1}{t}\ln||y_1(t)|| &= \lim_{t\to\infty} \frac{1}{t}\ln||y_1(0)|| + \lim_{t\to\infty} \frac{1}{t}\int_0^t [\gamma(\tau)\lambda_1 - \beta(\tau)]d\tau \\ &= 0 + \bar{\gamma}\lambda_1 - \bar{\beta} < 0, \end{aligned}$$

which implies $\lim_{t\to\infty} y_1(t) = 0$.

Assuming $\lim_{t\to\infty} y_v(t) = 0$ already proved, consider $y_{v+1}(t)$. We have

$$y_{v+1}(t) = y_{v+1}(0)\exp\left(\int_0^t k(\tau)d\tau\right) + \int_0^t \xi_{v+1}(a)\exp\left(\int_a^t k(\tau)d\tau\right)da$$

where $k(\tau) = \gamma(\tau)\lambda_{k_{v+1}} - \beta(\tau)$. From condition (2.7), there exists a sufficiently small $\epsilon > 0$ such that $(\bar{\gamma} + \epsilon)\lambda_1 - (\bar{\beta} + \epsilon) < 0$. Let $\varphi = -(\bar{\gamma} + \epsilon)\lambda_1 + (\bar{\beta} + \epsilon)$. One can see that $\varphi > 0$. Since the limits $\int_{t_0}^{t_0+t} \beta(\tau)d\tau$ and $\int_{t_0}^{t_0+t} \gamma(\tau)d\tau$ are uniform with respect to t_0, there exists $T > 0$ such that

$$\frac{1}{t-a}\int_a^t \beta(\tau)d\tau < \bar{\beta} + \epsilon, \; \frac{1}{t-a}\int_a^t \gamma(\tau)d\tau < \bar{\gamma} + \epsilon$$

hold for any a and t with $t - a > T$. Let $\mathcal{R}e(z)$ denote the real part of a complex number

z. Then, we have

$$\begin{aligned}\mathcal{R}e\left(\frac{1}{t-a}\int_a^t k(\tau)d\tau\right) &= \frac{1}{t-a}\int_a^t [\gamma(\tau)\mathcal{R}e(\lambda_{k_{v+1}}) - \beta(\tau)]d\tau \\ &\le \frac{1}{t-a}\int_a^t [\gamma(\tau)\lambda_1 - \beta(\tau)]d\tau \\ &\le (\bar{\gamma}+\epsilon)\lambda_1 - (\bar{\beta}+\epsilon) = -\varphi < 0\end{aligned}$$

for all t and a with $t-a>T$. This implies that the first term $y_{v+1}(0)\exp(\int_0^t k(\tau)d\tau)$ converges to zero. Let $M>0$ be a constant such that $\sup_{\tau,j}|\gamma(\tau)\lambda_j - \beta(\tau)| < M$. For the second term, we have

$$\begin{aligned}&\left|\int_0^t \xi_{v+1}(a)\exp\left(\int_a^t k(\tau)d\tau\right)da\right| \\ =& \left|\int_0^{t-T} \xi_{v+1}(a)\exp\left(\int_a^t k(\tau)d\tau\right)da + \int_{t-T}^t \xi_{v+1}(a)\exp\left(\int_a^t k(\tau)d\tau\right)da\right| \\ \le& \int_0^{t-T} |\xi_{v+1}(a)|\exp\left(-\varphi(t-a)\right)da + \int_{t-T}^t |\xi_{v+1}(a)|\exp(MT)da.\end{aligned}$$

In the case of $\xi_{v+1}(t)=0$, we immediately conclude that $\lim_{t\to\infty} y_{v+1}(t)=0$. Otherwise, according to the condition and the L'Hospital principle, we have

$$\begin{aligned}\lim_{t\to\infty}\int_0^{t-T}|\xi_{v+1}(a)|\exp\big[-\varphi(t-a)\big]da &= \lim_{t\to\infty}\frac{\int_0^{t-T}|\xi_{v+1}(a)|\exp(\varphi a)da}{\exp(\varphi t)} \\ &= \lim_{t\to\infty}\frac{|\xi_{v+1}(t-T)|\exp(\varphi(t-T))}{\varphi\exp(\varphi t)} \\ &= \lim_{t\to\infty}\frac{|y_v(t-T)\gamma(t-T)|\exp(-\varphi T))}{\varphi} = 0,\end{aligned}$$

due to the assumption $\lim_{t\to\infty} y_v(t)=0$. We also have

$$\lim_{t\to\infty}\int_{t-T}^t |\xi_{v+1}(a)|\exp(MT)da = 0.$$

Therefore, we can conclude $\lim_{t\to\infty} y_{v+1}(t)=0$.

Note that condition (2.8) implies that system (2.4) is unstable. Since system (2.4) is in fact the linearization system of Eq. (2.2), we can conclude that system (2.2) is unstable under condition (2.8). This completes the proof. □

If $\beta(t)$ and $\gamma(t)$ are ergodic stochastic processes, from the multiplicative ergodic theory of the random dynamical systems [Arnold 1998], we have the following result as a corollary of Theorem 2.

COROLLARY 1. (another succinct sufficient condition) *Suppose $\beta_v(t)=\beta(t)$ for all $v\in V$ (i.e., all nodes have the same cure capability) and $E(t)=E$ for any time t (i.e., topology does not change over time). Suppose $\{\beta(t)\}_{t\ge0}$ and $\{\gamma(t)\}_{t\ge0}$ are ergodic stochastic processes (i.e., $\beta(t)$ and $\gamma(t)$ are some random variables). Let $\mathsf{E}(\beta(0))$ and $\mathsf{E}(\gamma(0))$ be the expectations with respect to the stationary distributions of the respective* *ergodic stochastic process. Suppose the convergences*

$$\mathsf{E}(\beta(0)) = \frac{1}{t}\int_{t_0}^{t_0+t} \beta(\tau)d\tau,\ \mathsf{E}(\gamma(0)) = \frac{1}{t}\int_{t_0}^{t_0+t} \gamma(\tau)d\tau,$$

are both uniform with respect to t_0 almost surely. *If*

$$\lambda_1 < \frac{\mathsf{E}(\beta(0))}{\mathsf{E}(\gamma(0))},$$

the spreading will die out almost surely regardless of the initial infection configuration; if

$$\lambda_1 > \frac{\mathsf{E}(\beta(0))}{\mathsf{E}(\gamma(0))},$$

the spreading will not die out in some initial infection configurations.

Discussion. The state-of-the-art sufficient condition for the dying out of virus spreading in an arbitrary network is $\lambda_1 < \frac{\beta}{\gamma}$, which was given in [Chakrabarti et al. 2008]. In the setting of [Chakrabarti et al. 2008], the parameters satisfy that $\beta_v(t) = \beta$ for all $v \in V$ and all t, and $\gamma(t) = \gamma$ for all t. As such, their result is clearly a special case of the above Corollary 1 (note that it is guaranteed that $\mathsf{E}(\gamma(0)) \neq 0$), and thus of the above Theorem 2.

2.3 Adaptive control in the scenario of fully-adaptive scenario

In the semi-adaptive defense scenario investigated above, we assumed that the parameters $\gamma(t)$ and $\beta_v(t)$ are given. What if they are not given? In what follows we investigate a representative scenario, where the defender is not given $\gamma(t)$ but can observe $i_v(t)$. Specifically, we consider two sufficient conditions of adaptive control: one under which the virus spreading will die out (Section 2.3.1), and another under which the virus spreading will not die out but will be contained to a desired level of infection (Section 2.3.2).

2.3.1 *Sufficient condition under which the virus spreading dies out under adaptive control.* The question we ask is: How should the defender adjust the defense, namely how $\beta_v(t)$ should depend upon $i_v(t)$, so that the virus spreading will die out? We assume for concreteness that $\beta_v(0) = 0$ for all $v \in V$; this accounts for the worst-case scenario.

THEOREM 3. (characterization of adaptive control strategy under which the virus spreading will die out) *Suppose without loss of generality* $\beta_v(0) = 0$ *for all* $v \in V$. *If*

$$\frac{d\beta_v(t)}{dt} = \rho i_v(t),$$

where ρ *is an (almost) arbitrary positive constant, then the virus spreading will die out regardless of the initial infection configuration.*

PROOF. Define a candidate Lyapunov function with respect to the infectious probabilities $i = [i_1, \cdots, i_n]^\top$ and the cure capabilities $\tilde{\beta} = [\beta_1, \cdots, \beta_n]^\top$:

$$V(i, \tilde{\beta}) = \sum_{v=1}^{n} i_v(t) + \frac{1}{2\rho}\sum_{v=1}^{n}(\beta_v(t) - \beta_{0,v})^2.$$

Let $\beta_{0,v}$, $v = 1, \cdots, n$, be positive constants satisfying

$$\beta_{0,v} > (n-1)\sup_t \gamma(t) + 1,\ \forall\ v = 1, \cdots, n$$

owing to the fact that $1 \geq \gamma(t) \geq 0$. Due to inequality (2.5), differentiating $V(i,\tilde{\beta})$ gives

$$\begin{aligned}\frac{dV(i,\tilde{\beta})}{dt} &= \sum_{v=1}^{n}\left[1-\prod_{(u,v)\in E(t)}(1-\gamma(t)i_u(t))\right](1-i_v(t))-\sum_{v=1}^{n}\beta_v(t)i_v(t)\\ &\quad+\sum_{v=1}^{n}(\beta_v(t)-\beta_{0,v})i_v(t)\\ &\leq \sum_{v=1}^{n}\gamma(t)\sum_{u=1}^{n}a_{vu}(t)i_u(t)-\sum_{v=1}^{n}\beta_v(t)i_v(t)+\sum_{v=1}^{n}(\beta_v(t)-\beta_{0,v})i_v(t)\\ &\leq \sum_{v=1}^{n}\gamma(t)\sum_{u=1}^{n}a_{vu}(t)i_u(t)-\sum_{v=1}^{n}\beta_{0,v}i_v(t)\\ &\leq \gamma(t)(n-1)\sum_{u=1}^{n}i_u(t)-\sum_{v=1}^{n}\beta_{0,v}i_v(t)\\ &\leq -\sum_{v=1}^{n}i_v(t).\end{aligned}$$

According to the LaSalle principle [LaSalle 1960], the system converges to the largest invariant set $\{(i,\tilde{\beta}) : \sum_{v=1}^{n} i_v = 0\}$, which implies $\lim_{t\to\infty} i_v(t) = 0$ for all $v = 1,\cdots,n$. □

The above Theorem 3 has the following implications.

PROPOSITION 1. *We can bound from above the accumulated number of infected nodes in the long run (a node is counted multiple times if it is infected at multiple points in time) as follows:*

$$\int_0^{\infty}\sum_{v=1}^{n} i(\tau)d\tau \leq \frac{(n-1)^2\gamma_m}{\rho}+\frac{n-1}{\rho}\sqrt{\sum_{v=1}^{n} i_v(0)+\frac{(n-1)^3\gamma_m^2}{2\rho}} \tag{2.10}$$

where $\gamma_m = \sup_t \gamma(t)$ *and* $\beta_{0,m} = \min_v \beta_{0,v}$.

PROOF. From the proof of Theorem 3, we have

$$\frac{dV(i,B)}{dt} \leq [\gamma_m(n-1)-\beta_{0,m}]\sum_{v=1}^{n} i_v(t),$$

which implies

$$\int_0^{t}\sum_{v=1}^{n} i_v(\tau)d\tau \leq \frac{1}{\beta_{0,m}-\gamma_m(n-1)}[V(0)-V(t)] \leq \frac{1}{\beta_{0,m}-\gamma_m(n-1)}V(0).$$

If $\beta_v(0) = 0$ for all $v = 1,\cdots,n$ and all $\beta_{v,0}$ are the same and are thus denoted by β_0, we have the following estimation:

$$\int_0^{\infty}\sum_{v=1}^{n} i(\tau)d\tau \leq \frac{1}{\beta_0-\gamma_m(n-1)}\big[\sum_{v=1}^{n} i_v(0)+\frac{1}{2\rho}(n-1)\beta_0^2\big].$$

By setting

$$\beta_0 = a + \frac{\sqrt{\sum_{v=1}^{n} i_v(0) + 1/(2\rho)(n-1)^3\gamma_m^2}}{1/(2\rho)(n-1)},$$

we obtain the minimum of the right-hand side and thus complete the proof. □

Physical meanings of Proposition 1. The term at the left-hand side of inequality (2.10) captures, in addition to the aforementioned estimation of the total number of infected nodes in the network (counting repetition) over time, the convergence rate of the adaptive control strategy. This allows us to draw the following insights: (i) The larger ρ, the faster the virus spreading will die out; (ii) the larger degree of initial infection, the slower the virus spreading will die out; (iii) the larger edge infection probability γ, the slower the virus spreading will die out.

2.3.2 *Adaptive control under which the virus spreading will not die out but will be contained to a desired level.* In the above we have given some sufficient condition on adjusting the defense or $\beta_v(t)$ so that the virus spreading will die out. What if the required $\beta_v(t)$ cannot be achieved, meaning that we may not expect that the virus spreading die out? This is possible because the defense may not be as good as one may wish or because of budget limitation. In this case, we ask an alternative interesting question: What it takes so that $i_v(t)$ can converge or be contained to some pre-determined level of infection i_v^*?

THEOREM 4. (characterization of adaptive control strategy under which the virus spreading will be contained to a desired level of infection) *Consider the following variant of the master equation Eq. (2.2),*

$$\frac{di_v(t)}{dt} = \left[1 - \prod_{(u,v)\in E}[1-\gamma\cdot i_u(t)]\right](1-i_v(t)) - \beta_v(t)i_v(t) + w_v \qquad (2.11)$$

For any $1 \geq i_v^* > 0$, $v = 1, \cdots, n$, letting

$$\beta_v^* = \frac{[1-\prod_{(u,v)\in E}(1-\gamma\cdot i_u^*)](1-i_v^*)}{i_v^*},$$

if we use the following adaptive control strategy

$$\frac{d\beta_v(t)}{dt} = \rho[i_v(t) - i_v^*]i_v(t), \qquad w_v = \eta[i_v^* - i_v(t)],$$

where ρ is an (almost) arbitrary positive constant, and η is a positive constant with $\eta + \min_v \beta_v^* > 1 + (1-\gamma)\lambda_1$ where λ_1 is the largest (in modulus) eigenvalue of the adjacency matrix A, we have $\lim_{t\to\infty} i_v(t) = i_v^*$.

PROOF. From Perron-Frobenius theorem [Berman and Shaked-Monderer 2003], we have that there exist some positive constants $P_1, \cdots, P_n$ such that $P = \mathrm{diag}[P_1, \cdots, P_n]$ satisfies $[PA + A^\top P] \leq \lambda_1 P$. Consider the candidate Lyapunov function with respect to $i = [i_1, \cdots, i_n]^\top$ and $\tilde{\beta} = [\beta_1, \cdots, \beta_n]^\top$:

$$V(i, \tilde{\beta}) = \frac{1}{2}\sum_{v=1}^{n} P_v[i_v - i_v^*]^2 + \frac{1}{2\rho}\sum_{v=1}^{n} P_v(\beta_v - \beta_v^*)^2.$$

Differentiating $V(x, \tilde{\beta})$ gives

$$
\begin{aligned}
& \frac{dV(i,\tilde{\beta})}{dt} \\
= \; & \sum_{v=1}^{n} P_v[i_v(t) - i_v^*]\Big\{[1 - \prod_{(u,v)\in E}(1-\gamma\cdot i_u(t))](1-i_v(t)) - \beta_v(t)i_v(t) - \eta(i_v - i_v^*)\Big\} \\
& + \sum_{v=1}^{n} P_v(\beta_v(t) - \beta_v^*)(i_v(t) - i_v^*)i_v(t) \\
= \; & \sum_{v=1}^{n} P_v[i_v(t) - i_v^*]\Big\{\big[1 - \prod_{(u,v)\in E}(1-\gamma\cdot i_u(t))\big](1-i_v(t)) \\
& -\big[1 - \prod_{(u,v)\in E}(1-\gamma\cdot i_u^*)\big](1-i_v^*) - \beta_v(t)i_v(t) - \beta_v^* i_v^* + \eta(i_v - i_v^*)\Big\} \\
& + \sum_{v=1}^{n} P_v(\beta_v(t) + \beta_v^*)(i_v(t) - i_v^*)i_v(t).
\end{aligned}
$$

Note that

$$
\begin{aligned}
& \Big|[1 - \prod_{(u,v)\in E}(1-\gamma\cdot i_u(t))](1-i_v(t)) - [1 - \prod_{(u,v)\in E}(1-\gamma\cdot i_u^*)](1-i_v^*)\Big| \\
\le \; & \Big|(i_v(t) - i_v^*)\prod_{(u,v)\in E}(1-\gamma i_v(t))\Big| + \Big|(1-i_v^*)\big[\prod_{(u,v)\in E}(1-\gamma i_u(t)) - \prod_{(u,v)\in E}(1-\gamma i_u^*)]\Big| \\
\le \; & |i_v(t) - i_v^*| + \sum_{(u,v)\in E}|i_u(t) - i_u^*| \prod_{(u_1,v)\in E,\ u_1>u}|1-\gamma i_{u_1}^*| \prod_{(u_2,v)\in E,\ u_2<u}|1-\gamma i_{u_2}(t)| \\
\le \; & |i_v(t) - i_v^*| + \gamma \sum_{(u,v)\in E}|i_u(t) - i_u^*|.
\end{aligned}
$$

Thus, we have

$$
\begin{aligned}
\frac{dV(i,\tilde{\beta})}{dt} \le \; & \sum_{v=1}^{n} P_v|i_v(t) - i_v^*|^2 + \sum_{v=1}^{n}\sum_{u=1}^{n}\gamma P_v a_{vu}|i_v(t) - i_v^*||i_u(t) - i_u^*| \\
& - \sum_{v=1}^{n}\beta_v^* P_v(i_v(t) - i_v^*)^2 - \eta\sum_{v=1}^{n} P_v[i_v(t) - i_v^*]^2.
\end{aligned}
$$

Let $z(t) = [z_1(t), \cdots, z_n(t)]^\top$ with $z_v(t) = |i_v(t) - i_v^*|$. Since $\beta_v^* \ge 0$, we have

$$
\begin{aligned}
\frac{dV(i,\tilde{\beta})}{dt} \le \; & z^\top\big[P + \gamma(PA + A^\top P)/2 - \eta P - B^* P\big]z \\
\le \; & z^\top\big[1 + \gamma\lambda_1 - \eta - \min_v \beta_v^*\big]Pz < 0\ , \forall\ z \neq 0,
\end{aligned}
$$

where $B^* = \mathrm{diag}[\beta_1, \cdots, \beta_n]$. Due to the LaSalle principle [LaSalle 1960], we have that the system will converge to the largest invariant set in $\{(i, \tilde{\beta}) : \sum_{v=1}^{n}[i_v - i_v^*]^2 = 0\}$, which implies that $\lim_{t\to\infty} i_v(t) = i_v^*$ for all $v = 1, \cdots, n$. □

Discussion. The above theorem is quite general because of the term w_v in Eq. (2.11). If $w_v \neq 0$, the adaptive control strategy must be used with caution because we must guarantee that its value does have physical meanings. In general, $w_v \leq 0$ would be reasonable; in our simulation study (Section 3), we set $w_v = 0$ for simplicity. On the other hand, the theorem is necessarily based on the premise that $\gamma(t) = \gamma$, namely that $\gamma(t)$ does not vary with respect to time, because of the way β_v^* is defined. In our simulation study (Section 3), we will show that the result is quite robust, meaning that even if $\gamma(t)$ varies with respect to time (as we considered in Section 2.2), the result is still valid. This is very important because the fixed γ can be seen as, in a sense, the average of the unknown $\gamma(t)$ over time.

Similar to Proposition 1, we can have

PROPOSITION 2. *We have*

$$\int_0^\infty \sum_{v=1}^n |i_v(\tau) - i_v^*| d\tau \;\leq\; \frac{1}{\min_v P_v(\eta - 1 - \gamma\lambda_1)} \Big[\frac{1}{2}\sum_{v=1}^n P_i |i_v(0) - i_v^*|^2 + \frac{1}{2\rho}\sum_{v=1}^n |\beta_v(0) - \beta_v^*|^2\Big]. \tag{2.12}$$

Physical meanings of Proposition 2. The above proposition offers the following insights: The larger ρ, the larger η, the smaller λ_1, the smaller γ, the smaller differential between $i_v(0)$ and i_v^*, or the smaller the differential between $\beta_v(0)$ and β_v^*, the faster the virus spreading will die out.

3. SIMULATION STUDY

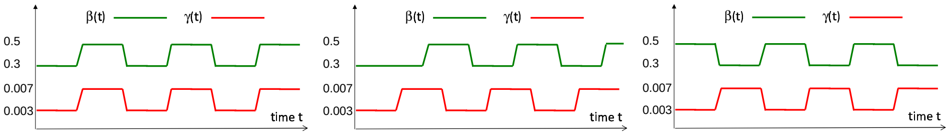
We conduct simulation to complement our analytic study for two purposes. First, we want to confirm our analytical results offered in Section 2. Second, we want to draw some relevant observations that are not offered by our analytic results. Such observations may guide future studies of analytic models (e.g., how to enhance them so that other useful insights may be obtained analytically).

As mentioned above, our model is very general because it accommodates dynamical graph topology $G(t) = (V, E(t))$. However, it's not clear at this stage how to appropriately define a physically meaningful way according to which the topology changes. Therefore we leave the full-fledged characterization (beyond what is implied by our analytical results) of the impact of dynamical topology to future work. We conducted simulations using both synthetic (regular, random, and power-law) graphs and a real network graph. Due to space limitation, here we report the simulation results in the latter case (but all the simulation results are consistent). The real network graph $G = (V, E)$ is based on the Oregon router views (available from `http://topology.eecs.umich.edu/data.html`), where $|V| = 11,461$ representing AS peers, $|E| = 32,730$ representing links between the AS peers. The largest eigenvalue of the corresponding adjacency matrix is $\lambda_1 = 75.2407$.

3.1 Methodology

Our simulation is conducted in an event-driven fashion. For the purpose of studying the dynamics under our adaptive control strategies, we need to measure $i_v(t)$, the probability that node $v \in V$ is infected at time t. This parameter can be obtained in a tedious way (i.e., by conducting for example 100 simulation runs in parallel, rather than in sequence, because we need to count the number of times each node v is infected at each time step).

A much more simpler way however is to use our model formula Eq. (2.2) to compute $i_v(t)$ instead, as long as the model is accurate. To confirm the accuracy of the model, we compare it with simulation. For simplicity, we let both $\beta(t)$ and $\gamma(t)$ be some periodic functions with period T, which means that both attack and defense vary with respect to time. We consider three settings: $\beta(t)$ and $\gamma(t)$ being synchronous, asynchronous, or anti-synchronous because we want to observe whether, and if so to what extent, the degree of (a)synchrony has an impact on the outcome. Specifically, we consider two sets of parameters: $\beta(t) \in \{0.3, 0.5\}$ and $\gamma(t) \in \{0.003, 0.007\}$ according to the functions shown in Figure 2; $\beta(t) \in \{0.005, 0.015\}$ and $\gamma(t) \in \{0.003, 0.007\}$ in the same fashion. In the asynchronous case, we let $\beta_v(t)$ is $T/4$ behind $\gamma_v(t)$ because cure often comes after attack is identified. To draw insights into whether the period T has an impact on the outcome, we consider $T = 8,\ 16$, respectively. In any case, it is clear that $\beta(t)$ is an implicit function of $\gamma(t)$.



(a) synchronous setting (b) asynchronous setting (c) anti-synchronous setting

Fig. 2. Examples of synchronous, asynchronous, and counter-synchronous $\beta(t)$ and $\gamma(t)$

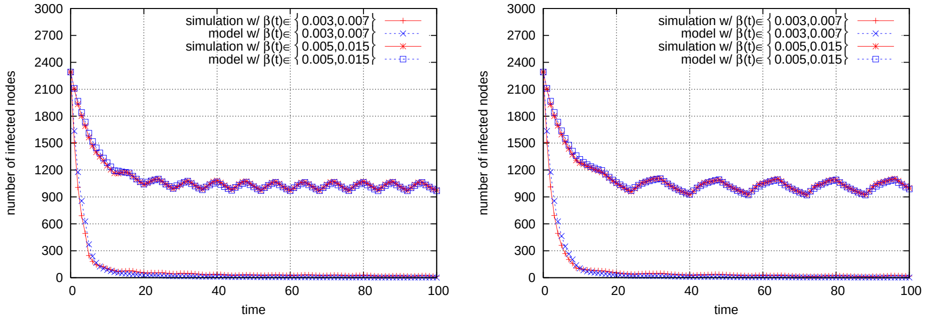
Figure 3 plots the curves obtained by simulation and by model computing in the case of both $\beta_v(t)$ and $\gamma_v(t)$ have period $T = 8$ and $T = 16$ (as shown in Figure 2). In each graph, we let the virus initially infect 2,292 or 20% vertices that are randomly selected; note that the degree of initial infection does not impact whether the virus spreading will die out or not. Since the model computing and simulation results (obtained as the average of 50 simulation runs) match almost perfectly no matter the virus spreading will die out or not, we will use simulation and model computing interchangeably. Since the same phenomenon applies to both cases of $T = 8$ and $T = 16$, in what follows we only report the case of $T = 8$. Again, the accuracy result allows us to obtain $i_v(t)$ via model computing in the process of confirming the analytical results of our adaptive control strategies.

3.2 Confirmation of the sufficient conditions in the semi-adaptive scenario

In this section, we use the aforementioned Oregon graph to confirm our analytical results presented in Section 2.2. We confirm Theorem 2 and Corollary 1 because they offer succinct sufficient conditions under which the virus spreading dies out.

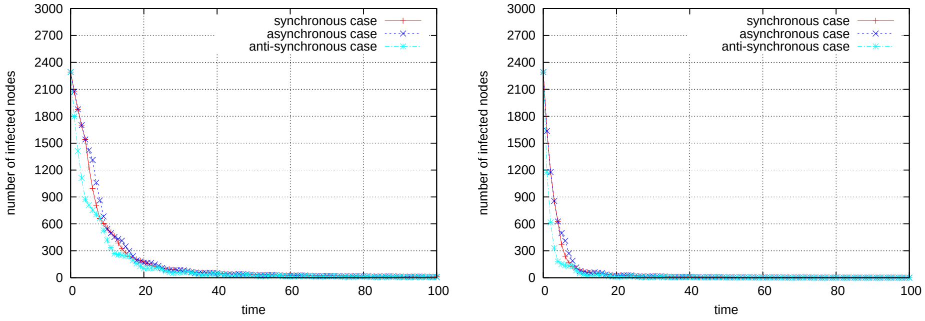
3.2.1 *Confirmation of Theorem 2.* For the graph, we let the virus initially infect 2,292 or 20% randomly selected nodes, and consider three cases between the model's input parameter — synchrony, asynchrony, and anti-synchrony as illustrated in Figure 2. Since Theorem 2 has two parts, we confirm them respectively.

Case 1: Confirmation of the sufficient condition under which the virus spreading will die out. We consider two sets of parameters: (i) $\beta(t) \in \{0.3, 0.5\}$ and $\gamma(t) \in \{0.003, 0.007\}$; (ii) $\beta(t) \in \{0.1, 0.22\}$ and $\gamma(t) \in \{0.001, 0.003\}$. Both functions, $\beta(t)$ and $\gamma(t)$, have period $T = 8$. Both parameter sets satisfy the sufficient condition of Theorem 2, namely $\lambda_1 < \bar{\beta}/\bar{\gamma}$, which means that the virus spreading will die out. Figure 4 plots



(a) T=8 (b) T=16

Fig. 3. Model accuracy

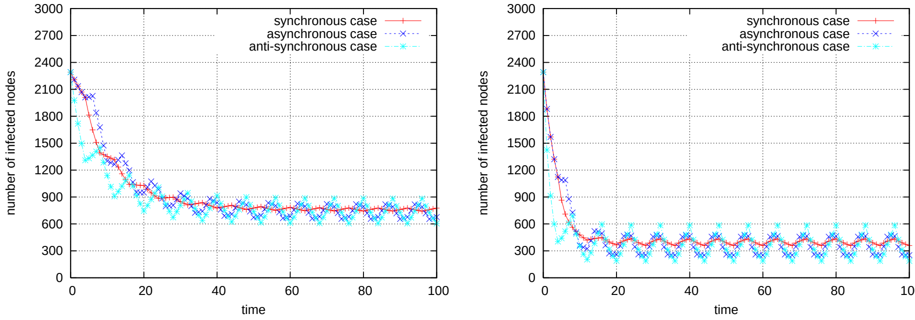


(a) Parameter set (i) (b) Parameter set (ii)

Fig. 4. Confirmation of sufficient condition under which virus spreading dies out ($T = 8$)

the dynamics of the numbers of infected nodes with respect to time. From Figure 4 we can draw the following observations. First, the virus spreading does die out at about the 50th and 25th step, respectively, which confirms the sufficient condition under which the virus spreading will die out. It is an interesting future work to quantitatively characterize how the speed of convergence (i.e., dying out) depends upon functions $\beta(t)$ and $\gamma(t)$. Second, it is counter-intuitive and interesting that the virus spreading is somewhat more effectively defended against in the anti-synchronous case than in the synchronous case, which is in turn more effectively defended against than in the asynchronous case. More studies are needed in order to explain this phenomenon. Third, the curves are convex, meaning that cure is more effective in the early stage of the attack-defense dynamics than in the later stage. For example, it takes a shorter period of time to reduce the infection from 2,292 nodes to 100 nodes than to reduce the infection from 100 nodes to zero nodes (i.e., dying out).

Case 2: Confirmation of the condition under which the virus spreading may not die out. We consider two sets of parameters: (i) $\beta(t) \in \{0.2, 0.4\}$ and $\gamma(t) \in \{0.003, 0.007\}$; (ii) $\beta(t) \in \{0.05, 0.15\}$ and $\gamma(t) \in \{0.001, 0.003\}$. Both functions, $\beta(t)$ and $\gamma(t)$, have period $T = 8$. Both parameter sets do not satisfy the sufficient condition of Theorem 2 because $\lambda_1 > \bar{\beta}/\bar{\gamma}$, which means that the virus spreading does not die out in some initial infection configurations.



(a) Parameter set (i)

(b) Parameter set (ii)

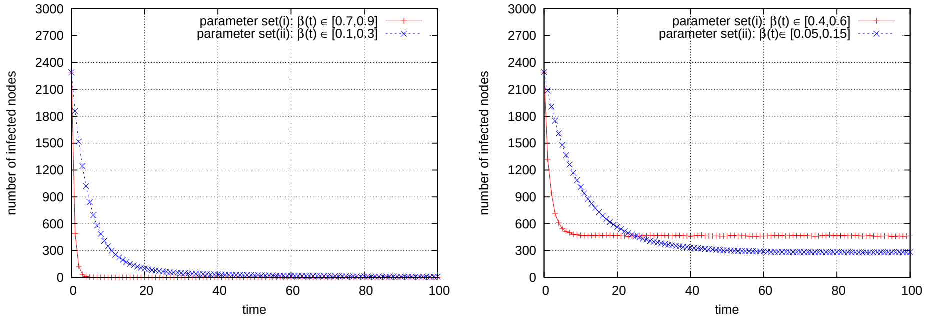
Fig. 5. Confirmation of virus spreading not dying out ($T = 8$)

Figure 5 plots the dynamics, from which we draw the following observations. First, the virus spreading does not die out, which confirms Theorem 2. Second, all the curves exhibit periodic behaviors, but the extent of oscillation in the case of anti-synchrony is more significant than in the case of asynchrony, which in turn is more significant than in the case of synchrony. This means that the degree of synchrony between $\beta(t)$ and $\gamma(t)$ will impact the outcome when the virus spreading does not die out. Third, comparing Figures 5(a) and 5(b), we observe that, under the same synchrony, the outcome will depend on functions $\beta(t)$ and $\gamma(t)$. More studies are needed to characterize these dependence relationships.

3.2.2 *Confirmation of Corollary 1.* Corollary 1 gives an even more succinct sufficient condition under which the virus spreading will die out. For the graph, we let the virus initially infect 2,291 or 20% randomly selected nodes. For the case the sufficient condition in Corollary 1 is satisfied, we consider two sets of parameters that are uniformly chosen at random from certain intervals: (i) $\beta(t) \in [0.7, 0.9]$ and $\gamma(t) \in [0.006, 0.0014]$; (ii) $\beta(t) \in [0.1, 0.3]$ and $\gamma(t) \in [0.0015, 0.0035]$. In each parameter setting, the sufficient condition stated in Corollary 1 is satisfied, namely $\lambda_1 < \mathsf{E}(\beta(0))/\mathsf{E}(\gamma(0))$, which means that the virus spreading will die out.

For the case the sufficient condition in Corollary 1 is not satisfied, we consider two sets of parameters that are uniformly chosen at random from certain intervals: (i) $\beta(t) \in [0.4, 0.6]$ and $\gamma(t) \in [0.006, 0.014]$; (ii) $\beta(t) \in [0.05, 0.15]$ and $\gamma(t) \in [0.0015, 0.0035]$. In each parameter setting, the sufficient condition stated in Corollary 1 is not satisfied because $\lambda_1 > \mathsf{E}(\beta(0))/\mathsf{E}(\gamma(0))$, and thus the analytical result says that the virus spreading does not die out in some initial infection configurations.

Figure 6(a) plots the dynamics of the number of infected nodes with respect to time when the sufficient condition is satisfied. From it we can draw the following observations. First, the virus spreading does die out, as predicted by Corollary 1. Second, the larger the $\beta(t)$, the more effective the defense against the virus spreading. Third, all the curves are convex, meaning that it takes a shorter period of time to significantly reduce the number



(a) The sufficient condition is satisfied

(b) The sufficient condition is not satisfied

Fig. 6. Confirmation of Corollary 1 ($T = 8$)

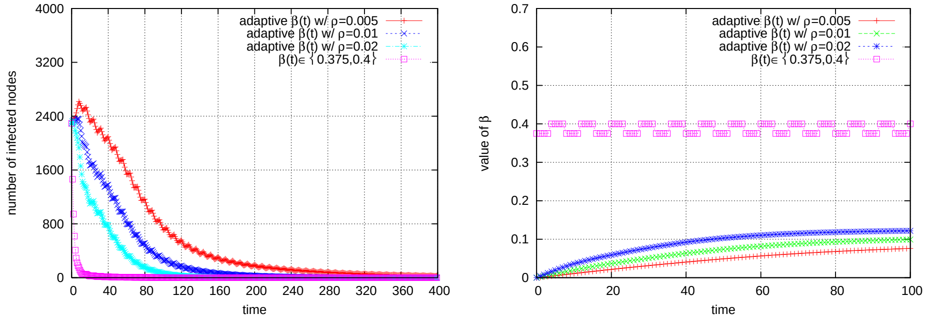
of infected nodes (e.g., from 2,291 to 50) than to making the virus spreading die out (e.g., from 50 to zero).

Figure 6(b) plots the dynamics of the number of infected nodes with respect to time when the sufficient condition is not satisfied. From it we can draw the following observations. First, the virus spreading does not die out, which confirms Corollary 1. Second, the larger (in a stochastic sense) the $\beta(t)$, the earlier the system will converge to the steady state. However, the ultimate degree of infection does not depend on $\beta(t)$, but rather on $\mathsf{E}(\beta(0))/\mathsf{E}(\gamma(0))$, which means that $\mathsf{E}(\beta(0))/\mathsf{E}(\gamma(0))$ may be used as an indicator of steady-state infection when the virus spreading does not die out. It is an interesting future work to rigorously characterize this phenomenon.

3.3 Confirmation of the controllability in the fully-adaptive scenario

3.3.1 *Confirmation of Theorem 3.* Theorem 3 states that even if we do not know $\gamma(t)$ but we may be able to observe $i_v(t)$ and may be able to adjust the defense as needed, following its control strategy will cause the dying out of the virus spreading. To compare the effects of adaptive control and semi-adaptive control, in our simulation study, we also used the periodical functions $\beta(t) \in \{0.375, 0.40\}$ and $\gamma(t) \in \{0.003, 0.007\}$ with period $T = 8$ as illustrated in Figure 2. These parameters satisfy the sufficient condition in Theorem 2, which means that the virus spreading will die out as we discussed above. To ensure comparability, we also let the virus initially infect 2,291 or 20% randomly selected nodes. As mentioned before, since the anti-synchronous defense is somewhat more effective than the synchronous and asynchronous defenses, we will compare it with the outcome of the adaptive control strategy.

Figure 7(a) plots the dynamics of the number of infected nodes with respect to time in the following four cases: the adaptive control parameter $\rho = 0.005$; the adaptive control parameter $\rho = 0.01$; the adaptive control parameter $\rho = 0.02$; the comparison dynamics corresponding to the anti-synchronous case with $T = 8$ periodical function $\beta(t) \in \{0.375, 0.40\}$. We draw the following observations. First, ρ plays a crucial role in indicating the rate at which the virus spreading dies out. For example, for $\rho = 0.02$, it takes only about 80 steps to reduce the number of infected nodes from 2,291 to 160 (nevertheless it takes another 60 steps to kill the virus spreading, namely to reduce the number of infected nodes from 160 to zero); for $\rho = 0.01$, it takes about 130 steps to reduce the



(a) Fully-adaptive control

(b) Comparison of cure capabilities

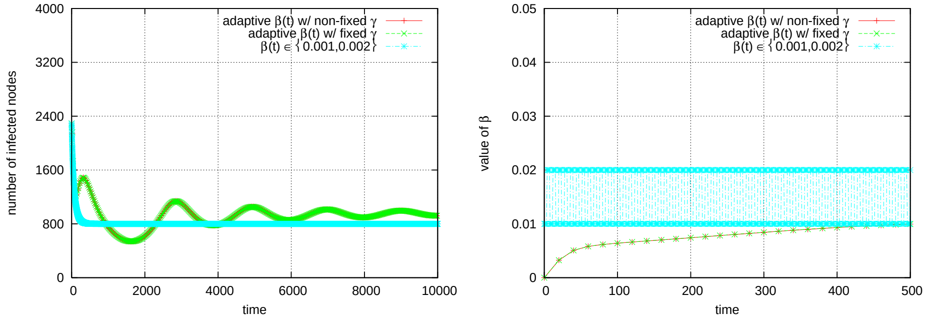
Fig. 7. Confirmation of semi-adaptive dying out vs. fully-adaptive control

number of infected nodes from 2,291 to 160 (nevertheless it takes about another 100 steps to kill the virus spreading, namely to reduce the number of infected nodes from 160 to zero). This also confirms the physical meanings of Proposition 1 discussed above.

Second, Figure 7(a) indicates that for all $\rho = 0.005$, $\rho = 0.01$ and $\rho = 0.02$, the fully-adaptive defenses are less effective than the semi-adaptive defense represented by $\beta(t) \in \{0.375, 0.40\}$. As we show in Figure 7(b), this is caused by the fact that the semi-adaptive $\beta(t)$ is much larger than the adaptive $\beta(t)$. This means that the sufficient condition in the semi-adaptive case, under which the virus spreading dies out, may be significantly beyond being necessary. In contrast, the fully-adaptive control strategy is much more "cost-effective" because larger $\beta(t)$ will likely cause a higher cost.

3.3.2 *Confirmation of Theorem 4.* Theorem 4 states that even if we do not know $\gamma(t)$ but we may be able to observe $i_v(t)$ and may be able to adjust the defense (but cannot kill the virus spreading), then following its control strategy will cause the containment of the virus spreading. In our simulation study, we used the periodical function $\gamma(t) \in \{0.0005, 0.001\}$ with period $T = 8$ similar to what was shown in Figure 2. This input parameter is not used in our adaptive control algorithm, rather it is merely for the purpose of comparison to the sufficient condition in Theorem 2, which requires the $\beta_v(t)$ satisfy certain property (for example, we use $\beta(t) \in \{0.01, 0.02\}$, meaning that the virus spreading does not die out as predicated), and the $\beta_v(t)$ derived from our adaptive control strategy. To ensure the comparability, we also let the virus initially infect 2,291 or 20% randomly selected nodes. As mentioned before, since the anti-synchronous defense is more effective than the synchronous and asynchronous defenses, we will compare it with the outcome of the adaptive control algorithm.

In our simulation we set $i^* = 0.1$ (i.e., we want to contain the degree of infection to 10%) and $u_v = 0$ (i.e., a special case of Theorem 4). Figure 8(a) plots the dynamics of the number of infected nodes with respect to time in the following three cases: the adaptive control parameter $\rho = 0.001$; the adaptive control parameter $\rho = 0.001$ with fixed $\bar{\gamma}(t) = (0.0005 + 0.001)/2 = 0.00075$ as specified in Theorem 4); the comparison dynamics corresponding to the anti-synchronous case with $T = 8$ periodical function $\beta(t) \in \{0.01, 0.02\}$. We draw the following observations. First, the control strategy does contain the infection to the pre-determined level of $i^* = 0.1$ or 10% infection. Moreover,



(a) Fully-adaptive control

(b) Comparison of cure capabilities

Fig. 8. Confirmation of the virus spreading containment via fully-adaptive control (the red-color curves are hidden behind the green-color curves)

the adaptive control strategy is robust because perturbation in γ does not fundamentally change the dynamics behavior. This also confirms the physical meanings of Proposition 2 discussed above.

Second, Figure 8(a) indicates that the adaptive defenses are slightly less effective in defending against the virus spreading than the defense of $T = 8$ periodical function $\beta(t) \in \{0.01, 0.02\}$. As we show in Figure 8(b), the adaptive control strategy can be much more “cost-effective” because it leads to significantly smaller $\beta(t)$.

4. CONCLUSION

We have presented a novel dynamical systems model for studying both semi-adaptive and fully-adaptive defenses against virus spreading. For semi-adaptive defense, we give general as well as succinct sufficient conditions under which the virus spreading will die out. For fully-adaptive defense, we characterize two adaptive control strategies under which the virus spreading will die out or will be contained to a desired level of infection. Our analytical results are confirmed with simulation study.

This paper brings a range of open questions for future research. In addition to those mentioned in the body of the paper, here are more examples: What are the necessary conditions under which the virus spreading will die out? What are the optimal adaptive control strategies?

Acknowledgement. We thank the anonymous reviewers for their useful comments, and Raj Boppana for helpful discussion on the simulation.

This work was supported in part by AFOSR, AFOSR MURI, ONR, and UTSA. The views and conclusions contained in the article are those of the authors and should not be interpreted as, in any sense, the official policies or endorsements of the government or the agencies.

REFERENCES

ANDERSON, R. AND MAY, R. 1991. *Infectious Diseases of Humans*. Oxford University Press.

ARNOLD, L. 1998. *Random Dynamical Systems*. Springer-Verlag.

BAILEY, N. 1975. *The Mathematical Theory of Infectious Diseases and Its Applications*. 2nd Edition. Griffin, London.

BERMAN, A. AND SHAKED-MONDERER, N. 2003. *Completely positive matrices*. World Scientific Publishing.

BHARGAVA, B., DILLEY, J., AND RIEDL, J. 1986. Raid: a robust and adaptable distributed system. In *ACM SIGOPS European Workshop*.

CHAKRABARTI, D., WANG, Y., WANG, C., LESKOVEC, J., AND FALOUTSOS, C. 2008. Epidemic thresholds in real networks. *ACM Trans. Inf. Syst. Secur. 10*, 4, 1–26.

GANESH, A., MASSOULIE, L., AND TOWSLEY, D. 2005. The effect of network topology on the spread of epidemics. In *Proceedings of IEEE Infocom 2005*.

HETHCOTE, H. 2000. The mathematics of infectious diseases. *SIAM Rev. 42*, 4, 599–653.

KEPHART, J. AND WHITE, S. 1991. Directed-graph epidemiological models of computer viruses. In *IEEE Symposium on Security and Privacy*. 343–361.

KEPHART, J. AND WHITE, S. 1993. Measuring and modeling computer virus prevalence. In *IEEE Symposium on Security and Privacy*. 2–15.

KERMACK, W. AND MCKENDRICK, A. 1927. A contribution to the mathematical theory of epidemics. *Proc. of Roy. Soc. Lond. A 115*, 700–721.

LASALLE, J. 1960. Some extensions of liapunov's second method. *IRE Trans. Circuit Theory 7*, 520–527.

MCKENDRICK, A. 1926. Applications of mathematics to medical problems. *Proc. of Edin. Math. Soceity 14*, 98–130.

OSELEDEC, V. 1968. A multiplicative ergodic theorem. characteristic ljapunov, exponents of dynamical systems. *English Translation. Trans. Moscow Math. Soc. 19*, 197–231.

PESIN, Y. B. 1977. Characteristic lyapunov exponents and smooth ergodic theory. *Russ. Math. Surv. 32*, 55–114.

WANG, Y., CHAKRABARTI, D., WANG, C., AND FALOUTSOS, C. 2003. Epidemic spreading in real networks: An eigenvalue viewpoint. In *Proc. of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS'03)*. 25–34.

ZOU, C., DUFFIELD, N., TOWSLEY, D., AND GONG, W. 2005. Adaptive defense against various network attacks. In *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'05)*. 69–75.

Received ????; accepted ????

A Stochastic Model of Multivirus Dynamics

Shouhuai Xu, Wenlian Lu, and Zhenxin Zhan

Abstract—Understanding the spreading dynamics of computer viruses (worms, attacks) is an important research problem, and has received much attention from the communities of both computer security and statistical physics. However, previous studies have mainly focused on *single-virus* spreading dynamics. In this paper, we study *multivirus* spreading dynamics, where multiple viruses attempt to infect computers while possibly combating against each other because, for example, they are controlled by multiple botmasters. Specifically, we propose and analyze a general model (and its two special cases) of multivirus spreading dynamics in arbitrary networks (i.e., we do not make any restriction on network topologies), where the viruses may or may not coreside on computers. Our model offers analytical results for addressing questions such as: What are the sufficient conditions (also known as epidemic thresholds) under which the multiple viruses will die out? What if some viruses can "rob" others? What characteristics does the multivirus epidemic dynamics exhibit when the viruses are (approximately) equally powerful? The analytical results make a fundamental connection between two types of factors: defense capability and network connectivity. This allows us to draw various insights that can be used to guide security defense.

Index Terms—Multiple virus dynamics, epidemic dynamics, epidemic threshold, complex networks, complex systems, cyber warfare model.

1 Introduction

To solve the problem of computer viruses (malware, worms, or bots), we need a set of approaches, ranging from legislation to technology. One technological approach, we call the *dynamical systems* approach, aims to understand the attack(-defense) dynamics so as to offer insights and guidance for defense from a whole-system perspective (i.e., by addressing *macroscopic* questions). This approach is complementary to the well established approaches such as access control and cryptography. Despite the attention that has been paid by communities including computer security and statistical physics, existing studies mainly focused on *single-virus* spreading dynamics. In this paper, we study *multivirus* spreading dynamics, where multiple viruses may be controlled by multiple attackers and thus may combat against each other. We have witnessed incidents of this type. For example, the `Welchia` worm tried to "kill" the Blaster worm by deleting the `Msblast.exe` file created by `W32.Blaster.Worm` [26]. Although such incidents are not exactly as full-fledged and sophisticated as the virus combats investigated in the paper, they can be seen as a prelude to (some of) the investigated scenarios that may soon become reality. As such, we need to investigate and understand the consequences of fighting viruses so as to get prepared by enhancing our body of knowledge.

1.1 Our Contributions

For multivirus spreading dynamics, there are two scenarios: the viruses spread independent of each other and thus the dynamics can be understood as a trivial extension of the single-virus dynamics; the viruses spread nonindependently and may further fight against each other. The present paper focuses on the latter scenario and its two special cases: 1) the viruses can coreside on computers, and 2) the viruses do not coreside on computers because the viruses can "rob" each other. Specifically, we make the following contributions.

First, we propose a novel continuous-time *nonlinear dynamical system* model for the spreading of *multiple* viruses in arbitrary networks (because any topology would be possible in practice). The general model is difficult to analyze, but we manage to present general sufficient conditions (also known as *epidemic thresholds*) under which all viruses will die out (one applies regardless of the degree of initial infection, and the other applies when the initial infection is light). We also present simplified versions of these sufficient conditions so as to make them easier to use.

Second, in order to gain deeper results, we further consider the aforementioned two special cases of the general model. While the special cases naturally inherit the sufficient conditions in the general model (which become more succinct in the special cases though), we additionally obtain the following results. In the special case, where multiple viruses do not coreside on computers, we investigate the combat between the viruses, namely that some viruses can "rob" other viruses (e.g., botmasters rob each other's bots). We analytically answer: 1) Under what conditions the weak viruses (i.e., viruses that can be robbed by, but cannot rob, others) will die out? 2) What are the outcomes when the viruses are (approximately) equally powerful? In the special case where multiple viruses may coreside on computers, we analytically answer: 1) Under what conditions some viruses will

- S. Xu and Z. Zhan are with the Department of Computer Science, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249. E-mail: shxu@cs.utsa.edu.
- W. Lu is with the School of Mathematical Sciences, Fudan University, 220 Handan Rd, Shanghai 200433, China. E-mail: wenlian@fudan.edu.cn.

Manuscript received 28 Apr. 2009; revised 14 June 2010; accepted 19 May 2011; published online 13 June 2011.
Recommended for acceptance by R. Sandhu.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2009-04-0059.
Digital Object Identifier no. 10.1109/TDSC.2011.33.

1545-5971/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

die out? 2) What are the outcomes when the viruses are (approximately) equally powerful in the setting of two viruses (the setting of three or more viruses is left as an open problem)? Finally, we conduct a simulation study, under the same model assumptions, to confirm the analytical results offered by the model.

Third, the analytical results make a fundamental connection between two types of factors: the defense capability and the connectivity of the network (via the largest eigenvalue of its adjacency matrix). Therefore, the analytical results not only deepen our understanding of multivirus spreading dynamics, but also offer insights and guidance for defense (e.g., how to tune the parameters by adjusting the defense mechanisms and/or how to tune a network so as to kill all or some of the viruses). The analytical results are often "succinct" (i.e., simple expressions of the parameters) enough to allow us to draw further insights by comparing them.

1.2 Related Work

There are mainly two approaches to understanding security properties from a whole-system perspective: epidemic-like and attack-graph-like. This paper falls into the epidemic-like approach.

Epidemic-like approach. This approach aims to characterize, among other things, sufficient conditions (epidemic thresholds) under which the spreading will die out. The study of computer virus dynamics was started by Kephart and White [19], [20], who adopted/adapted some *homogeneous* biological epidemic models [22], [21], [3], [2], [15]. Recent studies investigated heterogeneous models (see, e.g., [35], [4], [25], [30], [31], [29], [10]), which are interesting not only for their own sake but also because of their practical potential (because any topology is possible). These studies were mainly conducted by the statistical physics community with an emphasis on power law networks (see, e.g., [25], [30], [31], [29]). Very recently, computer scientists investigated spreading in arbitrary networks [34], [13], [10] (rather than just power law topologies). This paper moves a significant step beyond them by considering *multiple* viruses spreading in arbitrary networks, where the viruses may combat against each other as well.

Attack-graph-like approach. In an *attack graph*, a node is the state of a network and an arc depicts a step in an attack that exploits vulnerabilities uncovered by vulnerability scanners (cf. [33], [18], [1] and their follow-ons). This approach aims to identify, among other things, 1) attack paths between a starting node or a set of starting nodes and some target node, and 2) guidance for security hardening [27], [24]. Recently, Bursztein and Goubault-Larrecq [8], [7], [6], [9] introduced the novel concept of *anticipation game*, which aims to introduce game theory into attack graphs so as to prove that a safety property holds and further to answer questions about strategic objectives (e.g., what is the most effective patching strategy?).

On the relationship between the two approaches. At the current stage of our knowledge, the two approaches are incompatible. Here, we discuss some informal observations, while pointing out that it is an important future work to precisely characterize the relationship (e.g., how can we unify them into a more general framework?). First, the two approaches share some common goals. 1) Both approaches take a whole-system perspective, rather than a component perspective. For example, the vulnerabilities used in the attack-graph-like approach can be used as an input for constructing the system graph used in the epidemic-like approach. 2) Both approaches are useful for "what if" analysis as we show in this paper. Second, the two approaches can answer different kinds of questions under different premises. 1) Attack-graph-like approach often assumes known vulnerabilities that can be obtained using scanners, and thus cannot deal with zero-day exploits. Epidemic-like approach in principle can accommodate zero-day exploits, perhaps with the help of domain expert knowledge. 2) Attack-graph-like approach suffers from the state explosion problem [1], [11]; whereas, epidemic-like models can handle very large networks (e.g., hundreds of millions of nodes). 3) Epidemic-like approach can make a fundamental connection between the defense capability and the network connectivity. We do not know how to accommodate network connectivity into the attack-graph-like approach.

Outline. In Section 2, we present and explore a general model of multivirus dynamics. We investigate in Section 3 the special case that the viruses may fight against each other and do *not* coreside on computers, and in Section 4 the special case that the viruses may coreside on computers. In Section 5, we draw further insights from the analytical results. In Section 6, we report our simulation study. We conclude the paper in Section 7 with open problems.

2 General Multivirus Dynamics Model

2.1 The Master Equation

Parameters. Let $\mathcal{J}$ denote a set of viruses with $|\mathcal{J}| \geq 2$. Suppose viruses spread according to a finite network $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes or vertices and E is the set of edges or arcs. We call G the *system graph*, and denote by $A = [a_{vu}]$ its adjacency matrix, where $a_{vv} = 0$ and for all $v \neq u$, $a_{vu} = 1$ if and only if $(u, v) \in E$. This representation accommodates, and thus our results equally apply to, both directed and undirected graphs.

A system graph abstracts a networked system. A vertex in a system graph my represent a computer with some vulnerabilities, and an edge or arc captures that the exploitation of one vulnerability or the compromise of one computer could lead to the exploitation/compromise of another. A system graph may be obtained, for example, by combining the output of vulnerability scanners and systems configurations (in a fashion similar to [12], [28], [33], [18], [1], [11]), by analyzing the firewall isolation/filtering rules (e.g., which computer is allowed to directly access which other computers), or by using the concept of dependency graphs in a fashion similar to [6]. It is worthwhile to point out that our goal is to investigate the dynamics *without* making any assumption about the system graph, which certainly exists and can be arbitrary in practice.

The following table summarizes the main parameters and notations used throughout the paper.

$A = [a_{vu}]$	adjacency matrix of system graph $G = (V, E)$ where $\|V\| = n$, $a_{vu} = 1$ if and only if $(u, v) \in E$, and $a_{vv} = 0$ for all $v \in V$.
$\bar{d}$	the maximum node (in-)degree in $G = (V, E)$, $\bar{d} = \max_{v \in V}\{\sum_{u \in V} a_{vu}\}$
$\lambda_1, \ldots, \lambda_n$	eigenvalues of adjacency matrix A with λ_1 being the largest (in modulus)
$\bar{\lambda}_1, \ldots, \bar{\lambda}_H$	the geometrically distinct eigenvalues of adjacency matrix A
$\mathcal{J}$	the set of multiple viruses
$\mathcal{R}e(z)$	the real part of a complex number z
$\lambda(M)$	the set of eigenvalues of a square matrix M
$\mathrm{diag}[x_1, \cdots, x_n]$	the diagonal matrix of the given diagonal elements
$\\|\cdot\\|$	the norm of vector or matrix with $\\|\cdot\\|_2$ denoting the 2-norm
$\mathbb{R}$	the set of real numbers

The following table summarizes the main notations used in the general model (Section 2) and their simplified versions (if applicable) used in its special cases (Sections 3 and 4). The simplification is for the purpose of more succinct representations of the results and easier recognition of the models in which the results are obtained.

$\mathbf{j}, \mathbf{k}, \mathbf{l}, \mathbf{m}$	subsets of $\mathcal{J}$ with $\mathbf{m} = \emptyset$ $(I, J, L \subseteq \mathcal{J}$ are used in Section IV)
$\vartheta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}$	the probability/capability that a k-infected node u, $(u, v) \in E$, causes a j-infected node v to become l-infected; in Section III, $\vartheta^{\{j\}}_{\emptyset,\{j\}}$ is simplified as γ_j and $\vartheta^{\{j\}}_{\{\ell\},\{j\}}$ is simplified as $\vartheta_{\ell,j}$; in Section IV, $\vartheta^{I\cup\{k\}}_{I,\{k\}}$ is simplified as $\vartheta_{I,k}$ where $I \subseteq \mathcal{J}$
$\delta^{\mathbf{l}}_{v,\mathbf{j},\mathbf{k}}(t)$	the probability at time t that a j-infected node v becomes l-infected because of its k-infected neighbors; in Section III, $\delta^{\{j\}}_{v,\emptyset,\{j\}}$ is simplified as $\delta_{v,j}$ and $\delta^{\{j\}}_{v,\{\ell\},\{j\}}$ is simplified as $\theta_{v,\ell,j}$; in Section IV, $\delta^{I\cup\{k\}}_{v,I,\{k\}}$ is simplified as $\delta_{v,I,k}$
$\beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}$	the probability/capability that a j-infected node becomes l-infected because of the curing of viruses belonging to $\mathbf{k} = \mathbf{j} \setminus \mathbf{l}$; in Section III, $\beta^{\emptyset}_{\{j\},\{j\}}$ is simplified as β_j; in Section IV, $\beta^{I\setminus\{j\}}_{I,\{j\}}$ is simplified as $\beta_{I,j}$
relation $\mathbf{I}$	$(\mathbf{j}, \mathbf{k}, \mathbf{l}) \in \mathbf{I}$ represents the state transition corresponding to $\delta^{\mathbf{l}}_{v,\mathbf{j},\mathbf{k}}$
relation $\mathbf{R}$	$(\mathbf{j}, \mathbf{k}, \mathbf{l}) \in \mathbf{R}$ represents the state transition corresponding to $\beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}$
$i_{v,\mathbf{j}}(t)$	the probability node $v \in V$ is j-infected at time t, where $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m}\}$; in Section III, $i_{v,\{j\}}(t)$ is simplified as $i_{v,j}$

The state transition diagram. We consider a continuous-time model. At any time $t \geq 0$, a node $v \in V$ is in one of the following m states: $\mathbf{1}, \ldots, \mathbf{m} - \mathbf{1}, \mathbf{m} = \emptyset$, where the first $m - 1$ sets are nonempty, distinct subsets of $\mathcal{J}$, and m may or may not be equal to $2^{|\mathcal{J}|}$. A node $v \in V$ is susceptible or m-infected if it is not infected (i.e., secure but vulnerable), and j-infected if it is infected by viruses belonging to $\mathbf{j}$, where $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m} - \mathbf{1}\}$. We use "infected" and "infectious" interchangeably. Each node $v \in V$ changes its state according to a directed *state transition diagram*, which should *not* be confused with the system graph $G = (V, E)$. (Fig. 1 is the general state transition diagram for the case of $|\mathcal{J}| = 2$.)

The vertices of a state transition diagram are the aforementioned states $\mathbf{1}, \ldots, \mathbf{m}$. To simplify presentation, we define:

- Infection-map $\mathbf{I}$: $(\mathbf{j}, \mathbf{k}, \mathbf{l}) \in \mathbf{I}$ is the transition of node v's state from $\mathbf{j}$ to $\mathbf{l}$ because of its neighboring nodes u in state $\mathbf{k}$, where $(u, v) \in E$. The transition probability at time t is

$$\delta^{\mathbf{l}}_{v,\mathbf{j},\mathbf{k}}(t) = 1 - \prod_{(u,v)\in E} \left(1 - \vartheta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}} i_{u,\mathbf{k}}(t)\right), \tag{2.1}$$

where $\vartheta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}$ is the probability/capability of a *single* k-infected neighbor u causing that v's state changes from $\mathbf{j}$ to $\mathbf{l}$. Note that possibly $\vartheta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}} = 0$ (e.g., when $\mathbf{k} = \mathbf{m}$) and thus $\delta^{\mathbf{l}}_{v,\mathbf{j},\mathbf{k}}(t) = 0$. Note also that in the derivation of (2.1), we assumed that the infection events and the curing events are independent.

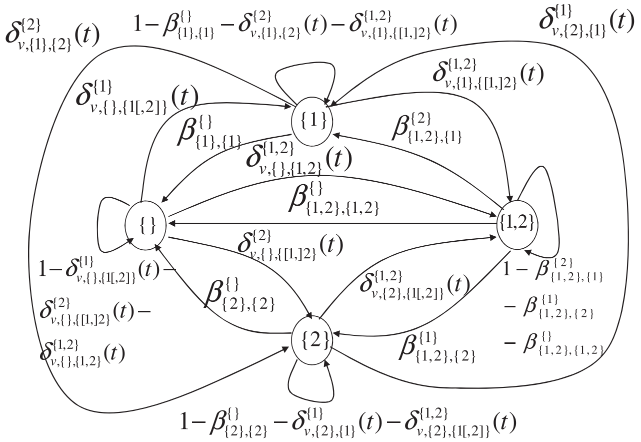

$\delta^{\{2\}}_{v,\{1\},\{2\}}(t)$
$1-\beta^{\{\}}_{\{1\},\{1\}} - \delta^{\{2\}}_{v,\{1\},\{2\}}(t) - \delta^{\{1,2\}}_{v,\{1\},\{[1,]2\}}(t)$
$\delta^{\{1\}}_{v,\{2\},\{1\}}(t)$
$\delta^{\{1\}}_{v,\{\},\{1[,2]\}}(t)$
$\delta^{\{1,2\}}_{v,\{1\},\{[1,]2\}}(t)$
$\{1\}$
$\beta^{\{\}}_{\{1\},\{1\}}$
$\beta^{\{2\}}_{\{1,2\},\{1\}}$
$\delta^{\{1,2\}}_{v,\{\},\{1,2\}}(t)$
$\beta^{\{\}}_{\{1,2\},\{1,2\}}$
$\{\}$
$\{1,2\}$
$\delta^{\{2\}}_{v,\{\},\{[1,]2\}}(t)$
$1-\delta^{\{1\}}_{v,\{\},\{1[,2]\}}(t)-$
$\delta^{\{2\}}_{v,\{\},\{[1,]2\}}(t)-$
$\delta^{\{1,2\}}_{v,\{\},\{1,2\}}(t)$
$\beta^{\{\}}_{\{2\},\{2\}}$
$\delta^{\{1,2\}}_{v,\{2\},\{1[,2]\}}(t)$
$1-\beta^{\{2\}}_{\{1,2\},\{1\}}$
$-\beta^{\{1\}}_{\{1,2\},\{2\}}$
$-\beta^{\{\}}_{\{1,2\},\{1,2\}}$
$\{2\}$
$\beta^{\{1\}}_{\{1,2\},\{2\}}$
$1-\beta^{\{\}}_{\{2\},\{2\}} - \delta^{\{1\}}_{v,\{2\},\{1\}}(t) - \delta^{\{1,2\}}_{v,\{2\},\{1[,2]\}}(t)$

Fig. 1. State transition diagram of $v \in V$ with $|\mathcal{J}| = 2$. We can define $\mathbf{m} = \{\}$, $\mathbf{1} = \{1\}$, $\mathbf{2} = \{2\}$, and $\mathbf{3} = \{1, 2\}$. For the sake of generality, we use the notation []; for example, $\delta^{\{1,2\}}_{v,\{1\},\{[1,]2\}}$ is short for "$\delta^{\{1,2\}}_{v,\{1\},\{2\}}$ or $\delta^{\{1,2\}}_{v,\{1\},\{1,2\}}$," meaning that the transition from state $\{1\}$ to state $\{1, 2\}$ may be caused by the neighboring nodes that were infected by virus 2 or by viruses 1 and 2. By setting certain transition probabilities to zero, we get special models that will be investigated for deeper results.

- Recovery-map $\mathbf{R}$: $(\mathbf{j}, \mathbf{k}, \mathbf{l}) \in \mathbf{R}$ is the transition of node v's state from $\mathbf{j}$ to $\mathbf{l}$ because of the curing of viruses belonging to $\mathbf{k} \subseteq \mathbf{j}$, where $\mathbf{j} \setminus \mathbf{k} = \mathbf{l}$. The transition probability is $\beta^{\mathbf{l}}_{\mathbf{j}}, \mathbf{k}$.

Define $\mathbf{P}^{\mathbf{l}}_{v,\mathbf{j},\mathbf{k}}(t)$, the probability that a node v's state transforms from $\mathbf{j}$ to $\mathbf{l}$ at time t as

$$\mathbf{P}^{\mathbf{l}}_{v,\mathbf{j},\mathbf{k}}(t) = \begin{cases} \beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}, & (\mathbf{j}, \mathbf{k}, \mathbf{l}) \in \mathbf{R}, \\ \delta^{\mathbf{l}}_{v,\mathbf{j},\mathbf{k}}(t), & (\mathbf{j}, \mathbf{k}, \mathbf{l}) \in \mathbf{I}. \end{cases}$$

Because an infected node may become susceptible again, our model is a susceptible-infectious-susceptible (SIS) one.

The master equation. We are concerned with $i_{v,\mathbf{j}}(t)$, the probability $v \in V$ is j-infected at time t where $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m}\}$. The master equation of the nonlinear dynamical system is

$$\begin{aligned} \frac{d}{dt} i_{v,\mathbf{j}}(t) = & \sum_{(\mathbf{l},\mathbf{k}):(\mathbf{l},\mathbf{k},\mathbf{j}) \in \mathbf{R} \bigcup \mathbf{I}} \mathbf{P}^{\mathbf{j}}_{v,\mathbf{l},\mathbf{k}} i_{v,\mathbf{l}}(t) \\ & - \sum_{(\mathbf{k}',\mathbf{l}'):(\mathbf{j},\mathbf{k}',\mathbf{l}') \in \mathbf{I} \bigcup \mathbf{R}} \mathbf{P}^{\mathbf{l}'}_{v,\mathbf{j},\mathbf{k}'} i_{v,\mathbf{j}}(t), \end{aligned} \tag{2.2}$$

for $v \in V$ and $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m}\}$. We notice that

$$\sum_{\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m}\}} i_{v,\mathbf{j}}(t) = 1 \text{ for } t \geq 0.$$

2.2 Preliminaries

We will need the following lemmas, whose notations may be treated as independent of the main body of the paper. Lemmas 1 and 2 are available from standard textbooks, but may be less known when compared with Gershgorin's disc theorem [16], the Perron-Frobenius theorem [5], and Grönwall's inequality [14], which will be used in our analysis as well.

Lemma 1 (Theorem 2.5.3, pp. 114, [17]). *Let $C = (c_{ij}) \in \mathbb{R}^{n,n}$ be a nonsingular matrix with $c_{ij} \leq 0$, $i, j = 1, \ldots, n$, and $i \neq j$. Then the following statements are equivalent.*

- *C is an M-matrix, meaning that all elements of C^{-1} are nonnegative.*
- *The real part of every eigenvalue of C is positive.*
- *There is a positive definite diagonal matrix P such that PCP^{-1} is strictly column (or row) diagonally dominant.*
- *There is a positive definite diagonal matrix Q such that $QC + C^\top Q$ is positive definite.*

Lemma 2 (Theorem 4.2, pp. 100, [14]). *System $\frac{d}{dt}x(t) = Bx(t)$, with $x(t) \in \mathbb{R}^n$ and $B \in \mathbb{R}^{n,n}$, is asymptotically exponentially stable if and only if the real part of every eigenvalue of B is negative.*

Lemma 3. *Consider a vector variable $u(t) = [u_1, \ldots, u_n(t)]^\top \in \mathbb{R}^n$ satisfying*

$$\frac{d}{dt}x_i(t) \le \sum_{j=1}^{n} c_{ij}x_j(t), \forall i = 1, \ldots, n, t \ge 0,$$

where $c_{ij} \ge 0$ for all $i \ne j$. Furthermore, consider the following comparison system:

$$\frac{d}{dt}y_i(t) = \sum_{j=1}^{n} c_{ij}y_j(t), \quad \forall i = 1, \ldots, n, t \ge 0.$$

If $x_i(0) \le y_i(0)$ for all $i = 1, \ldots, n$, then $x_i(t) \le y_i(t)$ holds for all $i = 1, \ldots, n$ and $t \ge 0$.

Proof. Let i_0 be the index of the component which satisfies $x_{i_0}(t) = v_{i_0}(t)$ for the first time, denoted by t_0. We have

$$\left.\frac{d[x_{i_0}(t) - y_{i_0}(t)]}{dt}\right|_{t=t_0} = c_{i_0 i_0}[x_{i_0}(t_0) - y_{i_0}(t_0)] + \sum_{j \ne i_0} c_{i_0 j}[x_j(t_0) - y_j(t_0)] = \sum_{j \ne i_0} c_{i_0 j}[x_j(t_0) - y_j(t_0)] \le 0,$$

owing to $x_j(t_0) \le y_j(t_0)$ for all $j \ne i_0$. This implies that $x_{i_0}(t) - y_{i_0}(t)$ does not strictly increases at $t = t_0$, which proves the lemma. □

2.3 General Sufficient Conditions for the Dying Out of All Viruses

Theorem 4 (A general sufficient condition for all viruses to die out regardless of the degree of initial infection). *Let λ_1 be the largest (in modulus) eigenvalue of the adjacency matrix A of the system graph $G = (V, E)$. Let $\boldsymbol{\Theta} = [\mathbf{a_{kj}}]$, where $\mathbf{k}, \mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$ and*

$$\boldsymbol{a}_{\mathbf{kj}} = \sum_{\mathbf{l}} : \ (l, k, j) \in \mathbf{I}\vartheta^{\mathbf{j}}_{\mathbf{l,k}}(\textit{possibly some } \mathbf{a_{kj}}'s \textit{ equal to zero}).$$

Let $\mathbf{B} = [\mathbf{b_{kj}}]$, *where* $\mathbf{k}, \mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$

and $\mathbf{b_{kj}} = \beta^{\mathbf{j}}_{\mathbf{k,l}}$(possibly some $\mathbf{b}'_{\mathbf{kj}}s$ equal to zero). Let $\mathbf{B}^{'} = diag[\mathbf{b}^{'}_1, \ldots, b^{'}_{m-1}]$, where $\mathbf{b}^{'}_{\mathbf{j}} = \sum_{(\mathbf{k,l}):(\mathbf{j,k,l}) \in \mathbf{R}} \beta^{\mathbf{l}}_{\mathbf{j,k}}$ for $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$. If

$$\max\{\mathcal{R}e(\xi) : \xi \in \lambda(\lambda_1\boldsymbol{\Theta} + \mathbf{B} - \mathbf{B}^{'})\} < 0, \tag{2.3}$$

all viruses will die out no matter how many nodes are initially infected with whichever viruses.

Proof. First, we show that it can be derived from condition (2.3) that

$$\max\{\mathcal{R}e(z) : z \in \lambda(w\boldsymbol{\Theta} + \mathbf{B} - \mathbf{B}^{'}), w \in \lambda(A)\} < 0. \tag{2.4}$$

The Perron-Forbenius theorem [5] says that λ_1 is a real number. Using the first and second items in Lemma 1, (2.3) implies that $-\lambda_1\boldsymbol{\Theta} - \mathbf{B} + \mathbf{B}'$ is an M-matrix and that there is a positive definite diagonal matrix $D = \text{diag}[d_{\mathbf{1}}, \ldots, d_{\mathbf{m}}]$ such that $D[\lambda_1\boldsymbol{\Theta} + \mathbf{B} - \mathbf{B}']D^{-1}$ is strictly diagonal dominant thanks to the third item of Lemma 1. Therefore, $D[w\boldsymbol{\Theta} + \mathbf{B} - \mathbf{B}']D^{-1}$ is strictly diagonal dominant for every $w \in \lambda(A)$. This leads to (2.4) according to Lemma 1 again.

Second, from (2.1) and (2.2), we have for $v \in V$ and $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$:

$$\begin{aligned}\frac{d}{dt}i_{v,\mathbf{j}}(t) = & \sum_{(\mathbf{l,k}):(\mathbf{l,k,j}) \in \mathbf{I}} \delta^{\mathbf{j}}_{v,\mathbf{l,k}} i_{v,\mathbf{l}}(t) + \sum_{(\mathbf{k,l}):(\mathbf{k,l,j}) \in \mathbf{R}} \beta^{\mathbf{j}}_{\mathbf{k,l}} i_{v,\mathbf{k}}(t) \\ & - \sum_{(\mathbf{k',l'}):(\mathbf{j,k',l'}) \in \mathbf{I}} \delta^{\mathbf{l'}}_{v,\mathbf{j,k'}} i_{v,\mathbf{j}}(t) - \sum_{(\mathbf{k',l'}):(\mathbf{j,k',l'}) \in \mathbf{R}} \beta^{\mathbf{l'}}_{\mathbf{j,k'}} i_{v,\mathbf{j}}(t) \\ \le & \sum_{(\mathbf{l,k}):(\mathbf{l,k,j}) \in \mathbf{I}} \left(\sum_{u \in V} a_{vu}\vartheta^{\mathbf{j}}_{\mathbf{l,k}} i_{u,\mathbf{k}}(t)\right) \\ & + \sum_{(\mathbf{k,l}):(\mathbf{k,l,j}) \in \mathbf{R}} \beta^{\mathbf{j}}_{\mathbf{k,l}} i_{v,\mathbf{k}}(t) - \sum_{(\mathbf{k',l'}):(\mathbf{j,k',l'}) \in \mathbf{R}} \beta^{\mathbf{l'}}_{\mathbf{j,k'}} i_{v,\mathbf{j}}(t) \\ = & \sum_{u \in V, \mathbf{k}} a_{vu}\mathbf{a_{kj}} i_{u,\mathbf{k}}(t) + \sum_{\mathbf{k}} \mathbf{b_{kj}} i_{v,\mathbf{k}}(t) - \mathbf{b}^{'}_{\mathbf{j}} i_{v,\mathbf{j}}(t).\end{aligned}$$

Now, we define the following comparison system:

$$\frac{d}{dt}x_{v,\mathbf{j}}(t) = \sum_{u \in V, \mathbf{k}} a_{vu}\mathbf{a_{kj}} x_{u,\mathbf{k}}(t) + \sum_{\mathbf{k}} \mathbf{b_{k,j}} x_{v,\mathbf{k}}(t) - \mathbf{b}^{'}_{\mathbf{j}} x_{v,\mathbf{j}}(t),$$

with $x_{v,\mathbf{j}}(0) = i_{v,\mathbf{j}}(0)$ for $v \in V$ and $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$. Note that $i_{v,\mathbf{j}}(t) \le x_{v,\mathbf{j}}(t)$ for all $t \ge 0$. By letting $X(t) = [x_{v,\mathbf{j}}(t)]_{v \in V, \mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m}\}}$, we get the following compact form:

$$\frac{d}{dt}X(t) = AX(t)\boldsymbol{\Theta} + X(t)\mathbf{B} - X(t)\mathbf{B}'. \tag{2.5}$$

Let $\bar{\lambda}_1, \bar{\lambda}_2, \ldots, \bar{\lambda}_H$ be the geometrically distinct eigenvalues of A. Let $A = S^{-1}JS$ be the adjacency matrix A's Jordan decomposition, where J is its Jordan canonical form and in block-diagonal form

$$J = \begin{bmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_H \end{bmatrix}, \text{where } J_h = \begin{bmatrix} \bar{\lambda}_h & 1 & 0 & \cdots & 0 \\ 0 & \bar{\lambda}_h & 1 & \cdots & 0 \\ 0 & 0 & \bar{\lambda}_h & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \bar{\lambda}_h \end{bmatrix} \text{for } 1 \le h \le H.$$

Let $Y(t) = SX(t)$. Then (2.5) can be rewritten as

$$\frac{d}{dt}Y(t) = JY(t)\boldsymbol{\Theta} + Y(t)\mathbf{B} - Y(t)\mathbf{B}'. \tag{2.6}$$

Write $Y(t)$ in row as $Y(t) = [Y_1(t), Y_2(t), \ldots, Y_n(t)]^\top$, where $n = |V|$ and each row $Y_v(t) \in \mathbb{R}^{m-1}$. Thus, for each $Y_v(t)$, (2.6) can be component-wisely rewritten as

$$\frac{d}{dt}Y_v(t) = Y_v(t)(\lambda_1\mathbf{\Theta} + \mathbf{B} - \mathbf{B}') + \xi_v(t), v \in V,$$

where $\xi_v(t) = 0$ if $Y_v(t)$ is an eigenvector of A and $\xi_v(t) = Y_{v+1}(t)\mathbf{\Theta}$ otherwise. Lemma 2 and (2.4) imply $\lim_{t\to\infty} Y(t) = 0$ exponentially if $\xi_v(t) = 0$ for all $t \geq 0$. By induction, we can prove $\lim_{t\to\infty} Y(t) = 0$ for all v. This is equivalent to $\lim_{t\to\infty} X(t) = 0$ exponentially. From Lemma 3, we have $i_{v,\mathbf{j}}(t) \leq X_{v,\mathbf{j}}(t)$ for all v, $\mathbf{j}$, and t. So, we have $\lim_{t\to\infty} i_{v,\mathbf{j}}(t) = 0$ exponentially and complete the proof. □

Now, we present a corollary that is easier to use.

Corollary 5 (a simplified sufficient condition for all viruses to die out regardless of the degree of initial infection). *If for all* $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$, *it holds that*

$$\lambda_1 < \frac{\sum_{(\mathbf{k},\mathbf{l}):(\mathbf{j},\mathbf{k},\mathbf{l})\in\mathbf{R}} \beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}} - \sum_{(\mathbf{l},\mathbf{k}):(\mathbf{l},\mathbf{k},\mathbf{j})\in\mathbf{R}} \beta^{\mathbf{j}}_{\mathbf{l},\mathbf{k}}}{\sum_{(\mathbf{l},\mathbf{k}):(\mathbf{l},\mathbf{k},\mathbf{j})\in\mathbf{I}} \vartheta^{\mathbf{j}}_{\mathbf{l},\mathbf{k}}}, \tag{2.7}$$

all viruses will die out no matter how many nodes are initially infected with whichever viruses.

Proof. From condition (2.7), we have for all $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$

$$-\mathbf{b}'_{\mathbf{j}} + \sum_{\mathbf{k}}[\lambda_1\mathbf{a}_{\mathbf{jk}} + \mathbf{b}_{\mathbf{jk}}] < 0. \tag{2.8}$$

Let z be an arbitrary eigenvalue of matrix $\lambda_1\mathbf{\Theta} + \mathbf{B} - \mathbf{B}'$. The Gershgorin disc theorem [16] says $|z + \mathbf{b}'_{\mathbf{j}} - \lambda_1\mathbf{a}_{\mathbf{jj}} - \mathbf{b}_{\mathbf{jj}}| \leq \sum_{\mathbf{k}\neq\mathbf{j}}[\lambda_1\mathbf{a}_{\mathbf{jk}} + \mathbf{b}_{\mathbf{jk}}]$. Thus, $\mathcal{R}e(z) \leq -\mathbf{b}'_{\mathbf{j}} + \sum_{\mathbf{k}}[\lambda_1\mathbf{a}_{\mathrm{jk}} + b_{\mathrm{jk}}]$, where the right-hand side is less than zero because of condition (2.8), which implies $\mathcal{R}e(z) < 0$ for all $z \in \lambda(\lambda_1\mathbf{\Theta} + \mathbf{B} - \mathbf{B}')$. Via Theorem 4, we complete the proof. □

Theorem 4 and Corollary 5 say that when the parameters satisfy the respective conditions, all viruses will be killed no matter how many nodes are initially infected. Therefore, the conditions may be overdemanding when only some (but not many) nodes are initially infected (i.e., light initial infection). In what follows we give a sufficient condition under which the viruses die out in the case of light initial infection; this sufficient condition is close to being necessary.

Theorem 6 (a sufficient condition for all viruses to die out in the case of light initial infection). *Let* $\tilde{\mathbf{\Theta}} = [\tilde{\mathbf{a}}_{\mathbf{kj}}] \in \mathbb{R}^{m-1,m-1}$, *where* $\mathbf{k},\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$ *and* $\tilde{\mathbf{a}}_{\mathbf{kj}} = \vartheta^{\mathbf{j}}_{\mathbf{m},\mathbf{k}}$. *If*

$$\max\{\mathcal{R}e(z) : z \in \lambda(\lambda_1\tilde{\mathbf{\Theta}} + \mathbf{B} - \mathbf{B}^{'})\} < 0, \tag{2.9}$$

then all viruses will die out; if

$$\max\{\mathcal{R}e(z) : z \in \lambda(\lambda_1\tilde{\mathbf{\Theta}} + \mathbf{B} - \mathbf{B}^{'})\} < 0, \tag{2.10}$$

then the viruses do not die out.

Proof. For $v \in V$ and $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$, consider the following linear system near $i_{v,\mathbf{m}} = 1$ or equivalently $i_{v,\mathbf{j}} = 0$ for all $v \in N$ and $\mathbf{j} = \mathbf{1}, \ldots, \mathbf{m-1}$, by neglecting all terms with order higher than 1

$$\begin{aligned}\frac{d}{dt}x_{v,\mathbf{j}}(t) &= \sum_{u\in V,\mathbf{k}} a_{vu}\vartheta^{\mathbf{j}}_{\mathbf{m},\mathbf{k}}x_{u,\mathbf{k}}(t) + \sum_{(\mathbf{k},\mathbf{l}):(\mathbf{k},\mathbf{l},\mathbf{j})\in\mathbf{R}} \beta^{\mathbf{j}}_{\mathbf{k},\mathbf{l}}x_{v,\mathbf{k}}(t) \\ &\quad - \sum_{(\mathbf{k},\mathbf{l}):(\mathbf{j},\mathbf{k},\mathbf{l})\in\mathbf{R}} \beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}x_{v,\mathbf{j}}(t) \\ &= \sum_{u\in V,\mathbf{k}} a_{vu}\tilde{\mathbf{a}}_{\mathbf{kj}}x_{u,\mathbf{k}}(t) + \sum_{\mathbf{k}}\mathbf{b}_{\mathbf{kj}}x_{v,\mathbf{k}}(t) - \mathbf{b}'_{\mathbf{j}}x_{v,\mathbf{j}}(t).\end{aligned} \tag{2.11}$$

Let $X(t) = [x_{v,\mathbf{j}}(t)]_{v\in V,\mathbf{j}\in\{\mathbf{1},\ldots,\mathbf{m}\}}$. Then, (2.11) can be rewritten as

$$\frac{d}{dt}X(t) = AX(t)\tilde{\mathbf{\Theta}} + X(t)\mathbf{B} - X(t)\mathbf{B}'. \tag{2.12}$$

Reasoning in a fashion similar to the proof of Theorem 4, we can prove (2.9) is sufficient for the local stability of system (2.2).

Condition (2.10) says that the real part of some eigenvalue of linear system (2.11) is greater than zero. Noticing that (2.11) is the linearization of (2.2) and according to the stable manifold theorem [32], if the linearized system has at least one eigenvalue with a positive real part, then the original system is unstable. Thus, we conclude that the original system (2.2) is unstable surrounding $i_{v,\mathbf{j}} = 0$ for all $\mathbf{j} \subseteq \mathcal{J}$. We complete the proof. □

In a similar fashion, we have the following corollary:

Corollary 7 (a simplified sufficient condition for the viruses to die out when the initial infection is light). *If for all* $\mathbf{j} \in \{\mathbf{1}, \ldots, \mathbf{m-1}\}$,

$$\lambda_1 < \frac{\sum_{(\mathbf{k},\mathbf{l}):(\mathbf{j},\mathbf{k},\mathbf{l})\in\mathbf{R}} \beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}} - \sum_{(\mathbf{l},\mathbf{k}):(\mathbf{l},\mathbf{k},\mathbf{j})\in\mathbf{R}} \beta^{\mathbf{j}}_{\mathbf{l},\mathbf{k}}}{\sum_{(\mathbf{k}):(\mathbf{m},\mathbf{k},\mathbf{j})\in\mathbf{I}} \vartheta^{\mathbf{j}}_{\mathbf{m},\mathbf{k}}}, \tag{2.13}$$

then all viruses will die out in the presence of light initial infection.

The above analytical results allow us to draw:

Insight 8. *The above analytical results, especially Corollaries 5 and 7, can quantitatively guide the deployment/adjustment of defense mechanisms (e.g., decreasing* $\vartheta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}$ *by enforcing deeper packet inspection over network edges, and/or increasing* $\beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}}$ *by deploying more antivirus softwares of different vendors, and/or decreasing* λ_1 *by disabling/filtering some network edges) so as to satisfy the sufficient conditions and to kill the spreading of all viruses.*

3 Deeper Results in the Case of Disjoint Spreading

In this section, we consider the special case that the fighting viruses do not coreside on computers, which allows us to simplify some notations. Specifically, we say a node $v \in V$ is susceptible if v is not infected, and j-infected if v is infected by virus j where $j \in \mathcal{J}$; these states, respectively, correspond to $\emptyset$-infected and $\{j\}$-infected in the general model. We are still concerned with: $s_v(t)$, the probability node $v \in V$ is susceptible at time t, which corresponds to $i_{v,\emptyset}(t)$ in the general model; $i_{v,j}(t)$, the probability $v \in V$ is j-infected at time t where $j \in \mathcal{J}$, which corresponds to $i_{v,\{j\}}(t)$ in the general model. The invariant is $s_v(t) + \sum_{j=1}^{|\mathcal{J}|} i_{v,j}(t) = 1$. Other parameters are:

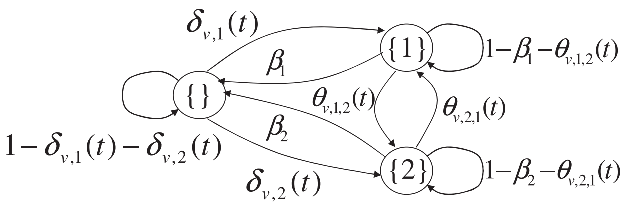


Fig. 2. A simplified state transition diagram of node $v \in V$ with $|\mathcal{J}| = 2$.

- β_j: The probability/capability a j-infected node becomes susceptible at any time, where $j \in \mathcal{J}$. Note that β_j corresponds to $\beta^{\emptyset}_{\{j\},\{j\}}$ in the general model. We may call the diagonal matrix $B = \mathrm{diag}[\beta_1, \ldots, \beta_{|\mathcal{J}|}]$ the "cure matrix."
- γ_j: The probability/capability a j-infected node u successfully infects a susceptible node v, where $(u, v) \in E$ and $j \in \mathcal{J}$. Note that γ_j corresponds to $\vartheta^{\{j\}}_{\emptyset,\{j\}}$ in the general model. We may call the diagonal matrix $\Gamma = \mathrm{diag}[\gamma_1, \ldots, \gamma_{|\mathcal{J}|}]$ the "infection matrix."
- $\vartheta_{\ell,j}$: The probability/capability that virus j successfully "robs" virus ℓ (i.e., a node v's state changes from ℓ-infected to j-infected), where $(u, v) \in E$, $\ell, j \in \mathcal{J}$, and $\ell \neq j$ (i.e., a virus does not rob itself). Note that $\vartheta_{\ell,j}$ corresponds to $\vartheta^{\{j\}}_{\{\ell\},\{j\}}$ in the general model. We may call $R = [\vartheta_{\ell,j}]^{|\mathcal{J}|}_{\ell,j=1}$ the "robbing matrix."
- $\delta_{v,j}(t)$: The probability a susceptible node v becomes j-infected at time t, with
$$\delta_{v,j}(t) = 1 - \prod_{(u,v)\in E} (1 - \gamma_j i_{u,j}(t)).$$
Note that $\delta_{v,j}(t)$ corresponds to $\delta^{\{j\}}_{v,\emptyset,\{j\}}(t)$ in the general model.
- $\theta_{v,\ell,j}(t)$: The probability an ℓ-infected v becomes j-infected at time t, with
$$\theta_{v,\ell,j}(t) = 1 - \prod_{(u,v)\in E} (1 - \vartheta_{\ell,j} i_{u,j}(t)).$$
Note that $\theta_{v,\ell,j}(t)$ corresponds to $\delta^{\{j\}}_{v,\{\ell\},\{j\}}(t)$ in the general model.

The state transition diagram depicted in Fig. 1 is now simplified as Fig. 2.

For a given state transition diagram, we define

$$\mathcal{N}_{in}(j) = \{\ell : \text{there is an arc from } \{\ell\} \text{ to } \{j\} \text{ in the state transition diagram}\},$$

$$\mathcal{N}_{out}(j) = \{\ell : \text{ there is an arc from } \{j\} \text{ to } \{\ell\} \text{ in the state transition diagram}\}.$$

In other words, $\ell \in \mathcal{N}_{in}(j)$ means virus j can rob virus ℓ, and $\ell \in \mathcal{N}_{out}(j)$ means virus ℓ can rob virus j. For example, corresponding to the state transition diagram in Fig. 2, we have $\mathcal{N}_{in}(1) = \mathcal{N}_{out}(1) = \{2\}$ and $\mathcal{N}_{in}(2) = \mathcal{N}_{out}(2) = \{1\}$ due to the symmetry that the viruses can rob each other. Then, the general master equation (2.2) becomes for $v \in V$

$$\begin{cases} \dfrac{d}{dt} s_v(t) = \left(-\sum_{j\in\mathcal{J}} \delta_{v,j}(t) \right) s_v(t) + \left(\sum_{j\in\mathcal{J}} \beta_j \right) i_{v,j}(t) \\ \dfrac{d}{dt} i_{v,j}(t) = \delta_{v,j}(t) s_v(t) + \left(\sum_{\ell\in\mathcal{N}_{in}(j)} \theta_{v,\ell,j}(t) \right) i_{v,\ell}(t) \\ \qquad - \left(\beta_j + \sum_{\ell\in\mathcal{N}_{out}(j)} \theta_{v,j,\ell}(t) \right) i_{v,j}(t), j \in \mathcal{J}. \end{cases} \tag{3.1}$$

3.1 More Succinct Sufficient Conditions for the Dying Out of All Viruses

Consider the special case where $\mathbf{l} = \{\ell\}$, $\mathbf{k} = \{k\}$, $\mathbf{j} = \{j\}$ in the general model. We have $\beta^{\mathbf{m}}_{\mathbf{j},\mathbf{m}} = \beta_j$, and $\beta^{\mathbf{l}}_{\mathbf{j},\mathbf{k}} = 0$ for all other cases; we have $\vartheta^{\mathbf{j}}_{\mathbf{m},\mathbf{j}} = \gamma_j$, $\vartheta^{\mathbf{j}}_{\mathbf{l},\mathbf{j}} = \vartheta_{\ell,j}$ for $\mathbf{l} \neq \mathbf{m}$ and $\mathbf{l} \neq \mathbf{j}$, and $\vartheta^{\mathbf{j}}_{\mathbf{l},\mathbf{k}} = 0$ otherwise. The following are special cases of Corollaries 5 and 7, respectively.

Corollary 9. *(more succinct sufficient condition for all viruses to die out regardless of the degree of initial infection) If for all*

$$j \in \mathcal{J}, \lambda_1 < \frac{\beta_j}{\gamma_j + \sum_{\ell\in\mathcal{J}} \vartheta_{\ell,j}}$$

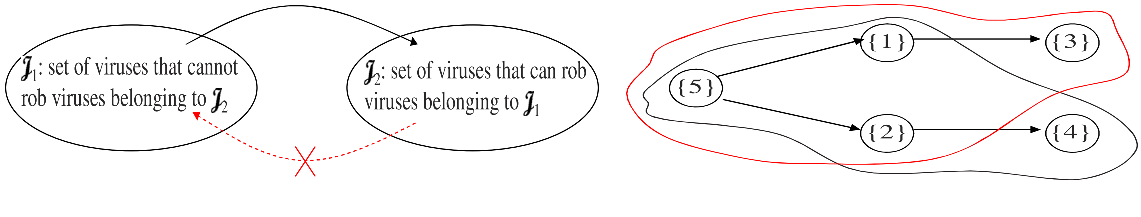
then all viruses will die out no matter how many nodes are initially infected.

Corollary 10 (more succinct sufficient condition for all viruses to die out in the case of light initial infection). *Suppose some, but not many, nodes are initially infected. If $\lambda_1 < \frac{\beta_j}{\gamma_j}$ holds $\forall j = 1, \ldots, |\mathcal{J}|$, then the viruses will die out. If $\exists j \in \{1, \ldots, |\mathcal{J}|\}$ such that $\lambda_1 > \frac{\beta_j}{\gamma_j}$, then the viruses do not die out.*

Insight 8 is naturally inherited by the above results.

3.2 What if Some Viruses Are More Powerful than Others?

Suppose the set of viruses $\mathcal{J}$ can be partitioned into two nonempty sets: $\mathcal{J}_1$, the set of weaker viruses, and $\mathcal{J}_2$, the set of stronger viruses, such that in the state transition diagram, there are no arcs starting at any $\{\ell\}$, $\ell \in \mathcal{J}_2$, and arriving at some $\{j\}$, $j \in \mathcal{J}_1$. In other words, a state transition diagram, after omitting the susceptible state, possesses the "weaker-stronger" structure depicted in Fig. 3a. Note that the viruses


$\mathcal{J}_1$: set of viruses that cannot rob viruses belonging to $\mathcal{J}_2$
$\mathcal{J}_2$: set of viruses that can rob viruses belonging to $\mathcal{J}_1$
$\{1\}$
$\{3\}$
$\{5\}$
$\{2\}$
$\{4\}$

(a) The weaker-stronger partitioning of viruses

(b) Non-uniqueness of maximal $\mathcal{J}_1$

Fig. 3. $\mathcal{J}$ may be partitioned, based on their robbing capabilities, into weaker viruses set $\mathcal{J}_1 \subset \mathcal{J}$ and stronger one $\mathcal{J}_2 = \mathcal{J} \setminus \mathcal{J}_1$.

belonging to $\mathcal{J}_1$ may be able to rob each other, but are unable to rob any virus belonging to $\mathcal{J}_2$. On the other hand, viruses belonging to $\mathcal{J}_2$ may be able to rob some viruses belonging to $\mathcal{J}_1$, and may additionally rob other viruses belonging to $\mathcal{J}_2$ as well.

Although there may be many ways to partition $\mathcal{J}$ while preserving the weaker-stronger structure, we may focus on the partitions where $|\mathcal{J}_1|$ is maximal because the weaker viruses may die before the stronger ones die. However, we should be aware that there may be multiple ways of partitioning while maximizing $|\mathcal{J}_1|$. For example, in the example shown in Fig. 3b with $\mathcal{J} = \{1,2,3,4,5\}$, virus 5 can be robbed by viruses 1 and 2; virus 1 can be robbed by virus 3; virus 2 can be robbed by virus 4. In this example, there are two ways of partitioning while maximizing $|\mathcal{J}_1|$: $\mathcal{J}_1 = \{1,2,3,5\}$ in one case (within red-color curve) and $\mathcal{J}_1 = \{1,2,4,5\}$ in the other (within blue-color curve). Therefore, we can define

$$\mathbb{J} = \max_{|\mathcal{J}'|}\{\mathcal{J}' \subset J : \nexists j', \ell, j' \in \mathcal{J}', \ell \in \mathcal{J} \setminus \mathcal{J}', \{\ell\} \text{ pointing to } \{j'\} \text{ in the state transition diagram}\}.$$

The following results concern the fate of the viruses belonging to any $\mathcal{J}_1 \in \mathbb{J}$ of interest.

Theorem 11 (sufficient condition for the dying out of weaker viruses). *If for all $j \in \mathcal{J}_1$,*

$$\lambda_1 < \frac{\beta_j}{\gamma_j + \sum_{\ell \in \mathcal{J}_1} \vartheta_{\ell,j}}, \tag{3.2}$$

then all viruses belonging to $\mathcal{J}_1$ will die out regardless of the number of nodes initially infected.

Proof. To prove the theorem, we also first prove the following result: If

$$\max\{\mathcal{R}](z) : z \in \lambda(\lambda_1(\Gamma_1 + R_{11}) - B_1)\} < 0, \tag{3.3}$$

then all viruses belonging to $\mathcal{J}_1$ will die out regardless of the degree of initial infection. Since condition (3.2) implies the condition (3.3), we complete the proof.

Now we prove that if condition (3.3) holds, then all viruses belonging to $\mathcal{J}_1$ will die out regardless of the degree of initial infection. For $j \in \mathcal{J}_1$, we have

$$\begin{aligned}\frac{d}{dt} i_{v,j}(t) &= \delta_{v,j}(t)s_v(t) + \sum_{\ell \in \mathcal{N}_{in}(j)} \theta_{v,\ell,j}(t) i_{v,\ell}(t) \\ &\quad - \left(\beta_j + \sum_{\ell \in \mathcal{N}_{out}(j)} \theta_{v,j,\ell}(t)\right) i_{v,j}(t) \\ &\le \sum_{u \in V} a_{vu} i_{u,j}(t)\gamma_j + \sum_{u \in V, \ell \in \mathcal{N}_{out}(j)} a_{vu}\vartheta_{\ell,j} i_{u,j}(t) i_{v,\ell}(t) \\ &\quad - \beta_j i_{v,j}(t) \\ &= \sum_{u \in V} a_{vu} i_{u,j}(t)\gamma_j + \sum_{u \in V}\sum_{\ell \in \mathcal{J}_1} a_{vu}\vartheta_{\ell,j} i_{u,\ell}(t) - \beta_j i_{v,j}(t),\end{aligned}$$

because $\vartheta_{\ell,j} = 0$ for all $j \in \mathcal{J}_1$ and $\ell \notin \mathcal{J}_1$. Write the robbing matrix R as $R = [\begin{smallmatrix} R_{11} & R_{12} \\ R_{21} & R_{22}\end{smallmatrix}]$, where R_{11} corresponds to the virus subset $\mathcal{J}_1$. According to the definition of $\mathcal{J}_1$, we have $R_{21} = 0$. Let Γ_1 and B_1 be submatrix of Γ and B corresponding to $\mathcal{J}_1$, respectively. Consider the following comparison system of $X_1(t) = [x_{v,j}(t)]_{v \in V, j \in \mathcal{J}_1}$: $\frac{d}{dt}X_1(t) = AX_1(t)(\Gamma_1 + R_{11}) - X_1(t)B_1$. We have $i_{v,j}(t) \le x_{v,j}(t)$ for all $v \in V$, $j \in \mathcal{J}_1$, and $t \ge 0$ when they take the same initial value, namely $i_{v,j}(0) = x_{v,j}(0)$, thanks to Lemma 3. Let $A = S^{-1}\mathcal{J}S$ be A's Jordan decomposition and $Y_1 = SX_1$, we have $\frac{d}{dt}Y_1 = \mathcal{J}Y_1(\Gamma_1 + R_{11}) - Y_1B_1$. Reasoning in a fashion similar to the proof of Theorem 4, we can conclude that under condition (3.3), $\max\mathcal{R}e(\lambda_v(\Gamma_1 + R_{11}) - B_1) < 0$ for all $v \in V$. This means $\lim_{t\to\infty} Y_1(t) = 0$. Therefore, $\lim_{t\to\infty} X_1(t) = 0$ and thus $\lim_{t\to\infty} i_{v,j}(t) = 0$ for all $v \in V$ and $j \in \mathcal{J}_1$. This completes the proof. □

Theorem 12 (sufficient condition for the dying out of weaker viruses in the case of light initial infection). *If $\forall j \in \mathcal{J}_1$, $\lambda_1 < \frac{\beta_j}{\gamma_j}$, then the viruses belonging to $\mathcal{J}_1$ will die out in the case of light initial infection with viruses belonging to $\mathcal{J}_1$.*

Proof. Note that for any $j \in \mathcal{J}_1$, if there exists ℓ with $\vartheta_{\ell,j} > 0$, then $\ell \in \mathcal{J}_1$. Considering only the viruses belonging to $\mathcal{J}_1$ in (3.1), and ignoring the terms of order higher than 2, we obtain for $j \in \mathcal{J}_1$

$$\begin{aligned}\frac{d}{dt} i_{v,j}(t) &= \sum_{u \in V} a_{vu}\gamma_j i_{u,j}(t)s_v(t) + \sum_{u \in V, \ell \notin \mathcal{J}_1} \vartheta_{\ell,j} a_{vu} i_{v,\ell}(t) i_{u,j}(t) \\ &\quad - \left(\beta_j + \sum_{\ell \in \mathcal{N}_{out}(j)} \theta_{v,j,\ell}(t)\right) i_{v,j}(t) \\ &\le \sum_{u \in V} a_{vu}\gamma_j i_{u,j}(t) - \beta_j i_{v,j}(t).\end{aligned}$$

Consider the following comparison system for $j \in \mathcal{J}_1$: $\frac{d}{dt}y_{v,j}(t) = \sum_{u \in V} a_{vu}\gamma_j y_{u,j}(t) - \beta_j y_{v,j}(t)$. Reasoning in a fashion similar to the proof of Theorem 6, we can show that under the given condition $\lambda_1 < \frac{\beta_j}{\gamma_j}$, $\lim_{t\to\infty} y_{v,j}(t) = 0$. Therefore, $i_{v,j}(t) = 0$ for $j \in \mathcal{J}_1$ are locally asymptotically stable. We complete the proof. □

The above analytical results allow us to draw:

Insight 13. *When it is impossible to kill all viruses, Theorems 11 and 12 offer guidelines for quantitatively tuning parameters (as mentioned above) to kill the weaker viruses.*

Because the weaker-stronger structure is not based on the curing-capabilities, but rather based on the robbing-capabilities, of the viruses, this offers an extra leverage to the defender:

Insight 14. *Even if we are unable to directly kill the weaker viruses, we may exploit the stronger viruses to help* alleviate *the weaker viruses when the stronger viruses are actually easier to kill.*

3.3 What if the Viruses Are (Approximately) Equally Powerful?

Here we address the question: What if the viruses are (approximately) equally powerful?

Theorem 15 (outcome of viruses being equally powerful). *Suppose for all j, $j' \in \mathcal{J}$ with $j \neq j'$, there exist β, γ, ϑ such that $\beta_j = \beta_{j'} = \beta$, $\gamma_j = \gamma_{j'} = \gamma$, and $\vartheta_{j,j'} = \vartheta_{j',j} = \vartheta$. If*

$$\lambda_1 < \frac{\beta}{\gamma + \vartheta}, \tag{3.4}$$

then $\forall v \in V$ and $j, j' \in \mathcal{J}$, $j \neq j'$, we have $\lim_{t\to\infty} |i_{v,j}(t) - i_{v,j'}(t)| = 0$.

Proof. Note that

$$\begin{aligned}\frac{d}{dt}i_{v,j}(t)-\frac{d}{dt}i_{v,j'}(t)&=(\delta_{v,j}(t)-\delta_{v,j'}(t))s_v(t)\\&+\sum_{\ell\in\mathcal{J}}[\theta_{v,\ell,j}(t)-\theta_{v,\ell,j'}(t)]i_{v,\ell}-\beta(i_{v,j}(t)-i_{v,j'}(t))\\&-\sum_{\ell\in\mathcal{J}}\theta_{v,j,\ell}(t)[i_{v,j}(t)-i_{v,j'}(t)].\end{aligned}$$

Since $\left|\delta_{v,j}(t)-\delta_{v,j'}(t)\right|\leq\sum_{u\in V}a_{vu}|i_{u,j}(t)-i_{u,j'}(t)|\gamma$ and $\left|\theta_{v,\ell,j}(t)-\theta_{v,\ell,j'}(t)\right|\leq\sum_{u\in V}a_{vu}\vartheta\left|i_{u,j}(t)-i_{u,j'}(t)\right|$, we have

$$\begin{aligned}\frac{d}{dt}\left|i_{v,j}(t)-i_{v,j'}(t)\right|&\leq(\gamma+\vartheta)\sum_{u\in V}a_{vu}\left|i_{u,j}(t)-i_{u,j'}(t)\right|\\&-\beta\left|i_{v,j}(t)-i_{v,j'}(t)\right|.\end{aligned}$$

Consider the following comparison system: $\frac{d}{dt}x(t)=[-\beta I_n+(\gamma+\vartheta)A]x(t)$, where I_n is the $n\times n$ identity matrix, and A is the adjacency matrix. Reasoning in a fashion similar to the proof of Theorem 9, we see that condition (3.4) implies that the comparison system is asymptotically stable, and thus $\lim_{t\to\infty}\left|i_{v,j}(t)-i_{v,j'}(t)\right|=0\ \forall v\in V,\ \forall j,j'\in\mathcal{J}$. □

Theorem 16 (outcome of viruses being approximately equally powerful). *Suppose for all ℓ, j, ℓ', $j'\in\mathcal{J}$ with $j\neq\ell$ and $j'\neq\ell'$, there exists some $\epsilon>0$ such that $|\beta_j-\beta_\ell|\leq\epsilon$, $|\gamma_j-\gamma_\ell|\leq\epsilon$, and $|\vartheta_{\ell,j}-\vartheta_{\ell',j'}|\leq\epsilon$. If*

$$\lambda_1<\frac{\beta_j}{\gamma_j+\vartheta_{\ell,j}},\tag{3.5}$$

holds for some ℓ,j, $\ell\neq j$, then there exists d^* independent of ϵ, such that $\overline{\lim}_{t\to\infty}|i_{v,j}(t)-i_{v,j'}(t)|\leq d^*\epsilon$ for all $v\in V$ and j, $j'\in\mathcal{J}$.

Proof. Note that, according to (3.1)

$$\begin{aligned}\frac{d}{dt}i_{v,j}(t)-\frac{d}{dt}i_{v,j'}(t)&=\big(\delta_{v,j}(t)-\delta_{v,j'}(t)\big)s_v(t)\\&+\sum_{\ell\in J}\big(\theta_{v,\ell,j}(t)-\theta_{v,\ell,j'}(t)\big)i_{v,\ell}(t)-\sum_{\ell\in J}\theta_{v,j,\ell}(t)(i_{v,j}(t)-i_{v,j'}(t))\\&-\beta_j\big(i_{v,j}(t)-i_{v,j'}(t)\big)+\big(\beta_{j'}-\beta\big)i_{v,j'}(t).\end{aligned}$$

Note also that

$$\begin{aligned}&\left|\delta_{v,j}(t)-\delta_{v,j'}(t)\right|\\&\leq\left|\prod_{(u,v)\in E}(1-\gamma_j i_{u,j'}(t))-\prod_{(u,v)\in E}(1-\gamma_j i_{u,j}(t))\right|\\&+\left|\prod_{(u,v)\in E}(1-\gamma_{j'} i_{u,j'}(t))-\prod_{(u,v)\in E}(1-\gamma_j i_{u,j'}(t))\right|\\&\leq\gamma_j\sum_{u\in V}a_{vu}\left|i_{u,j}(t)-i_{u,j'}(t)\right|+\sum_{u\in V}a_{vu}\left|\gamma_j-\gamma_{j'}\right|\\&\leq\gamma_j\sum_{u\in V}a_{vu}\left|i_{u,j}(t)-i_{u,j'}(t)\right|+\bar{d}\epsilon,\end{aligned}$$

where $\bar{d}$ is the largest node (in-)degree, namely $\bar{d}=\max_v\{|\{u:(u,v)\in E\}|\}$. Similarly

$$\left|\theta_{v,\ell,j}(t)-\theta_{v,\ell,j'}(t)\right|\leq\sum_{u\in V}a_{vu}\vartheta_{\ell,j}\left|i_{u,j}(t)-i_{u,j'}(t)\right|+\bar{d}\epsilon.$$

Therefore, we have

$$\begin{aligned}&\frac{d}{dt}\left|i_{v,j}(t)-i_{v,j'}(t)\right|\\&\leq\left|\delta_{v,j}(t)-\delta_{v,j'}(t)\right|+\sum_{\ell\in\mathcal{J}}\left|\theta_{v,\ell,j}(t)-\theta_{v,\ell,j'}(t)\right|\\&-\sum_{\ell\in\mathcal{J}}\theta_{v,j,\ell}(t)\left|i_{v,j}(t)-i_{v,j'}(t)\right|-\beta_j\left|i_{v,j}(t)-i_{v,j'}(t)\right|\\&+\left|\beta_{j'}-\beta\right|\leq(\gamma_j+\vartheta_{\ell,j})\sum_{u\in V}a_{vu}\left|i_{u,j}(t)-i_{u,j'}(t)\right|\\&-\beta_j\left|i_{v,j}(t)-i_{v,j'}(t)\right|+(|\mathcal{J}|+2)\bar{d}\epsilon.\end{aligned}$$

Let $\mathbf{1}=[1,1,\ldots,1]^\top$. Consider the following comparison system:

$$\frac{d}{dt}x(t)=[-\beta_j\mathrm{I}_n+(\gamma_j+\vartheta_{\ell,j})A]x(t)+d'\epsilon\mathbf{1},\tag{3.6}$$

for some given $\ell,j\in\mathcal{J}$, $\ell\neq j$, and $d'=\bar{d}(|\mathcal{J}|+2)$ which is clearly independent of ϵ. Let $W_j=-\beta_jI_n+(\gamma_j+\vartheta_{\ell,j})A$. From condition (3.5), we see that $-W_j$ is an M-matrix, and thus Lemma 1 says that there exists positive definite diagonal matrix Z_j such that $Z_jW_j+W_j^\top Z_j$ is negative definite. Let ρ_j^m and ϱ_j^m be the smallest (in modulus) eigenvalues of $1/2(Z_jW_j+W_j^\top Z_j)$ and Z_j, respectively. Then, we have

$$\begin{aligned}\frac{d}{dt}x^\top(t)Z_jx(t)&=x^\top(t)[Z_jW_j+W_j^\top Z_j]x(t)+2d'\epsilon x^\top(t)\mathbf{1}\\&\leq2\rho_j^mx^\top(t)x(t)+\varepsilon x^\top(t)x(t)+\frac{(d'\epsilon)^2n}{\varepsilon}\\&\leq\frac{2\rho_j^m+\varepsilon}{\|Z_j\|_2}x^\top(t)Z_jx(t)+\frac{(d'\epsilon)^2n}{\varepsilon},\end{aligned}$$

for all $-2\rho_j^m<\varepsilon<0$. Setting $\varepsilon=-\rho_j^m$ and using Grönwall's inequality [14], we have

$$\begin{aligned}x^\top(t)Z_jx(t)&\leq\exp\left(\frac{2\rho_j^m+\varepsilon}{\|Z_j\|_2}t\right)x^\top(0)Z_jx(0)\\&+\frac{(d'\epsilon)^2n}{\varepsilon}\frac{\|Z_j\|_2}{-2\rho_j^m-\varepsilon}\left[1-\exp\left(\frac{2\rho_j^m+\varepsilon}{\|Z_j\|_2}t\right)\right]\\&\leq\exp\left(\frac{\rho_j^m}{\|Z_j\|_2}t\right)x^\top(0)Z_jx(0)\\&+\frac{(d'\epsilon)^2\|Z_j\|_2n}{\left(\rho_j^m\right)^2}\left[1-\exp\left(\frac{\rho_j^m}{\|Z_j\|_2}t\right)\right].\end{aligned}$$

Therefore, we have

$$\overline{\lim}_{t\to\infty}\|x(t)\|_2^2\leq\overline{\lim}_{t\to\infty}\frac{x^\top(t)Z_jx(t)}{\varrho_j^m}\leq\frac{(d'\epsilon)^2n}{(\rho_j^m)^2}\frac{\|Z_j\|_2}{\varrho_j^m}.$$

One can see that condition (3.5) implies that the comparison system (3.6) satisfies $\overline{\lim}_{t\to\infty}\|x(t)\|_2\leq d^*\epsilon$ for some d^* independently of ϵ. This leads to $\overline{\lim}_{t\to\infty}\left|i_{v,j}(t)-i_{v,j'}(t)\right|\leq d^*\epsilon$ for all $v\in Z$, where d^* can be estimated as

$$d^*=\frac{d'}{\rho_j^m}\sqrt{\frac{\|Z_j\|_2n}{\varrho_j^m}}.$$

This completes the proof. □

The above analytical results allow us to draw:

Insight 17. *When the viruses can rob each other, if the viruses 1) have (approximately) equal attack power and (approximately) equal robbing power and 2) are (approximately) equally defended against, then they will have (approximately) the same fate, no matter how many nodes they initially infect and no matter they will die out or not. Moreover, if they die out, they will die out (asymptotically) at the same time.*

4 Deeper Results in the Case of Nondisjoint Spreading

Now we consider the other special case, where one computer gets incrementally infected (i.e., by one virus after another rather than concurrently) and the viruses on a computer get sequentially (rather than simultaneously) cured. We use capital letters such as I ($I \subseteq \mathcal{J}$) to represent states in the state transition diagram, and simplify other parameters as follows:

1. The infection-map $\mathbf{I}$ in the general model is now simplified as binary relation $(I, j) \in \mathbf{I}$ where $j \in \mathcal{J}$ and $j \notin I$, meaning that a node's state changes from I to $I \cup \{j\}$ because of infection by a $\{j\}$-infected node. We hereby denote the infection probability by $\vartheta_{I,j}$. The probability of node v's state changes from I to $I \cup \{j\}$ is simplified as
$$\mathbf{P}_{v,I,j}(t) = \delta_{v,I,j}(t) = 1 - \prod_{(u,v)\in E} (1 - \vartheta_{I,j} i_{u,\{j\}}(t)). \tag{4.1}$$
2. The recovery-map $\mathbf{R}$ in the general model is simplified as a binary relation $(I, j) \in \mathbf{R}$, where $j \in I$, meaning that a node's state changes from I to $I \setminus \{j\}$ because of curing. We hereby denote the cure probability by $\beta_{I,j}$.

The state transition diagram in Fig. 1 is simplified as the one in Fig. 4. The master equation (2.2) is now simplified as

$$\frac{d}{dt} i_{v,I}(t) = \sum_{(J,k)\in \mathbf{I}: J \bigcup \{k\}=I} \delta_{v,J,k} i_{v,J}(t) + \sum_{(K,\ell)\in \mathbf{R}: K\setminus\{\ell\}=I} \beta_{K,\ell} i_{v,K}(t) - \sum_{(I,k)\in \mathbf{I}} \delta_{v,I,k} i_{v,I}(t) - \sum_{(I,\ell)\in \mathbf{R}} \beta_{I,\ell} i_{v,I}(t), \tag{4.2}$$

where $v \in V$, $I \subseteq \mathcal{J}$, and $I \neq \emptyset$.

4.1 More Succinct Sufficient Conditions for the Dying Out of All Viruses

Because the present model is a special case of the general model, we can immediately obtain the following results as corollaries of 5 and 7, respectively.

Corollary 18 (sufficient condition for all viruses to die out regardless of the degree of initial infection). *If* $\forall I \subseteq \mathcal{J}$ *where*

$$I \neq \emptyset, \lambda_1 < \frac{\sum_{(I,j)\in \mathbf{R}} \beta_{I,j} - \sum_{(J,j)\in \mathbf{R}: J\setminus\{j\}=I} \beta_{J,j}}{\sum_{(K,k)\in \mathbf{I}: K \bigcup \{k\}=I} \vartheta_{K,k}},$$

where $J, K \subseteq \mathcal{J}$ *and* $j, k \in \mathcal{J}$, *then all viruses will die out no matter how many nodes are initially infected.*

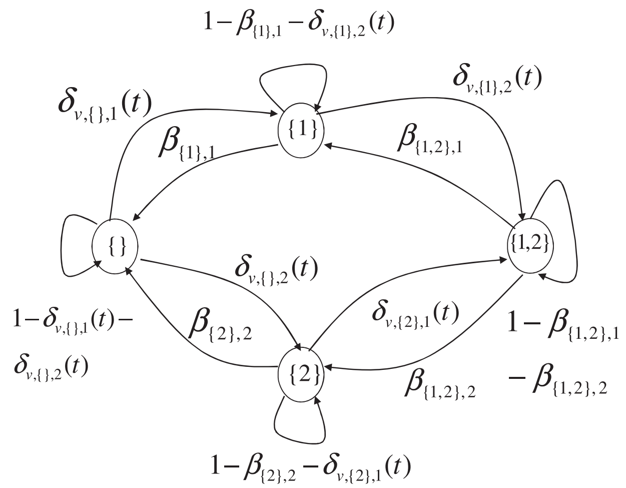


Fig. 4. Simplified state transition diagram of two viruses with nondisjoint spreading.

Corollary 19 (sufficient condition for all viruses to die out in the case of light initial infection). *All viruses will die out in the presence of light initial infection if: 1) for all* $J = \{j\}$ *where* $j \in \mathcal{J}$, *it holds that*

$$\lambda_1 < \frac{\beta_{\{j\},j} - \sum_{J=\{j,j'\}} \beta_{J,j'}}{\vartheta_{\{\},j}},$$

and 2) for all $I \subseteq \mathcal{J}$ *with* $|I| > 1$, *it holds that*

$$\sum_{(I,j)\in \mathbf{R}} \beta_{I,j} - \sum_{(J,j)\in\ \mathbf{R}: J-\{j\}=I} \beta_{J,j} > 0.$$

Note that in the latter case $\vartheta_{\{\},I} = 0$ *because* $|I| > 1$.

Insight 8 is also naturally inherited by the above results.

4.2 Under What Conditions Some Viruses Die Out?

Now we present sufficient conditions, under which a subset of viruses $J \subset \mathcal{J}$ will die out.

Theorem 20 (sufficient condition for the dying out of some viruses no matter how many nodes they initially infected). *For a set of viruses* $J \subset \mathcal{J}$, *if for every* $I \subseteq \mathcal{J}$ *with* $I \cap J \neq \emptyset$

$$\sum_{j\in I} \beta_{I,j} > \sum_{J'\setminus\{j'\}=I} \beta_{J',j'} + \lambda_1 \sum_{k\in J, K\cup\{k\}=I} \vartheta_{K,k} + \bar{d} \sum_{k\notin J, K\cup\{k\}=I} \vartheta_{K,k}, \tag{4.3}$$

where $\bar{d}$ *is the maximum node (in-)degree and* $k \notin K$, *then all viruses belonging to* J *will die out no matter how many nodes are initially infected.*

Proof. Note that, according to (4.1) for each $K \subset \mathcal{J}$ and $k \in \mathcal{J}$ with $k \notin K$, we have

$$\delta_{v,K,k}(t) i_{v,K}(t) \leq \sum_{u\in V} a_{vu} \vartheta_{K,k} i_{v,K}(t) i_{u,\{k\}}(t).$$

Furthermore, for each $K \subset I$ and $K \cup \{k\} = I$, we can make the following estimations:

$$\delta_{v,K,k}(t)i_{v,K}(t) \leq \begin{cases} \sum\limits_{u\in V} a_{vu}\vartheta_{K,k}i_{u,\{k\}}(t), & k \in J, \\ \bar{d}\vartheta_{K,k}i_{v,K}(t), & k \notin J. \end{cases}$$

Let $X(t) = [X_{v,I}(t)]$ with $v \in V$ and $I \bigcap \mathcal{J} \neq \emptyset$. Let $\Gamma = [\gamma_{KI}]$ with $K, I \in \{L \subseteq J : L \cap J \neq \emptyset\}$, where

$$\gamma_{KI} = \begin{cases} \vartheta_{I\setminus\{k\},k}, & K = \{k\} \subseteq I \cap J, (I \setminus \{k\}, k) \in \mathbf{I}, \\ 0, & otherwise. \end{cases}$$

Let $\Delta = [\Delta_{KI}]$ with $K, I \in \{L \subseteq J : L \cap J \neq \emptyset\}$, where

$$\Delta_{KI} = \begin{cases} \vartheta_{K,k}, & k \notin J, k \notin K, K \cup \{k\} = I, (K,k) \in \mathbf{I}, \\ 0, & otherwise. \end{cases}$$

Consider the following comparison system of the master equation (4.2)

$$\frac{d}{dt}X(t) = AX(t)\Gamma + \bar{d}X(t)\Delta + X(t)B - X(t)B',$$

with $B = [b_{KI}]$ and $B' = \mathrm{diag}[b'_I]$, where $K, I \in \{L \subseteq \mathcal{J} : L \cap J \neq \emptyset\}$ and

$$b_{KI} = \begin{cases} \beta_{K,j}, & K = I \cup \{j\}, (K,j) \in \mathbf{R}, \\ 0, & otherwise, \end{cases} \quad \text{and } b'_I = \sum_{(I,j)\in\mathbf{R}} \beta_{I,j}.$$

It can be seen that $X_{v,I}(t) \geq i_{v,I}(t)$ for all $t \geq 0$, $v \in N$, and $I \cap J \neq \emptyset$ as long as $X_{v,I}(0) \geq i_{v,I}(0)$. Consider the stability of the following component system:

$$\frac{d}{dt}y(t) = y(t)[\lambda_1\Gamma + \bar{d}\Delta + B - B'],$$

with $y(t)$ being n-dimensional row vector. Similar to the proof of Corollary 5, we conclude that condition (4.3) implies that the real parts of the eigenvalues of matrix $\lambda_1\Gamma + \bar{d}\Delta + B - B'$ are all negative. According to Lemma 2, we have $\lim_{t\to\infty} y(t) = 0$. By the same fashion of reasoning as in the proof of Theorem 4, we have $\lim_{t\to\infty} X(t) = 0$. According to Lemma 3, we have $\lim_{t\to\infty} i_{v,I}(t) = 0$ for all $I \bigcap J \neq \emptyset$. This completes the proof. □

Theorem 21 (sufficient condition for the dying out of some viruses that did not infect many nodes initially). *For a set of viruses $J \subset \mathcal{J}$, if for each I with $I \bigcap J \neq \emptyset$*

$$\sum_{j\in I} \beta_{I,j} > \sum_{J'\setminus\{j'\}=I} \beta_{J',j'} + \lambda_1 \sum_{k\in J, K\bigcap J=\emptyset, K\cup\{k\}=I} \vartheta_{K,k} + \bar{d} \sum_{k\notin J, K\cup\{k\}=I} \vartheta_{K,k}, \tag{4.4}$$

then all viruses belonging to J will die out when they did not initially infect many nodes.

Proof. Note that a linearization of the master equation (4.2) near $i_{v,I}(0) = 0$ for all $v \in N$ and $I \bigcap J \neq \emptyset$, neglecting all terms with order higher than 1, leads to

$$\delta_{v,K,j}(t)i_{v,K}(t) \begin{cases} \leq d_v\vartheta_{K,k}i_{v,K}(t), & k \notin J, K \bigcap J \neq \emptyset, \\ \leq \sum\limits_{u\in V} a_{vu}\vartheta_{K,j}i_{u,\{k\}}(t), & k \in J, K \bigcap J = \emptyset, \\ = 0, & otherwise. \end{cases}$$

Let $Z(t) = [Z_{v,I}(t)]$ with $v \in N$ and $I \cap J \neq \emptyset$. Let $\hat{\Gamma} = [\hat{\gamma}_{KI}]$ with $K, I \in \{L \subseteq \mathcal{J} : L \cap J \neq \emptyset\}$, where

$$\hat{\gamma}_{KI} = \begin{cases} \vartheta_{K',k}, & K{=}\{k\} \subset I, K'{=}I \setminus\{k\}, K' \bigcap J{=}\emptyset, (K',k){\in}\mathbf{I}, \\ 0, & otherwise. \end{cases}$$

Let $\hat{\Delta} = \mathrm{diag}[\hat{\Delta}_{KI}]$ with $K, I \in \{L \subseteq \mathcal{J} : L \cap J \neq \emptyset\}$, where

$$\hat{\Delta}_{KI} = \begin{cases} \vartheta_{K,k}, & K \cup \{k\} = I,\ k \notin J,\ (K,k) \in \mathbf{I}, \\ 0, & otherwise. \end{cases}$$

Define a comparison system to the linearized system near $i_{v,I}(0) = 0$ for all $v \in V$ and $I \bigcap J \neq \emptyset$

$$\frac{d}{dt}Z(t) = AZ(t)\hat{\Gamma} + \bar{d}Z(t)\hat{\Delta} + Z(t)B - Z(t)B',$$

where $B = [b_{KI}]$ and $B' = \mathrm{diag}[b'_I]$ with

$$b_{KI} = \begin{cases} \beta_{K,j}, & K = I \cup \{j\},\ (K,j) \in \mathbf{R}, \\ 0, & otherwise, \end{cases} \quad b'_I = \sum_{j\in I} \beta_{I,j}.$$

By the same fashion of reasoning as in the proof of Theorem 20, we can conclude the local stability of $i_{v,I} = 0$ for all $v \in N$ and $I \bigcap J \neq \emptyset$. □

The above analytical results allow us to draw:

Insight 22. *When the goal is to kill a subset of viruses (e.g., the damaging ones), Theorems 20 and 21 offer guidance for adjusting the defense (as mentioned above) toward the goal.*

4.3 What if the Viruses Are (Approximately) Equally Powerful?

When viruses can coreside on computers, it is much more challenging to analyze the outcome when the viruses are (approximately) equally powerful. Indeed, we are only able to analyze this for the special case of $|\mathcal{J}| = 2$ with respect to the state transition diagram shown in Fig. 4.

Theorem 23 (outcome of viruses being equally powerful). *Suppose $\vartheta_{\{\},1} = \vartheta_{\{\},2}$, $\vartheta_{\{1\},2} = \vartheta_{\{2\},1}$, $\beta_{\{1\},1} = \beta_{\{2\},2}$, and $\beta_{\{1,2\},1} = \beta_{\{1,2\},2}$. If $\lambda_1 < \frac{\beta_{\{1\},1}}{\vartheta_{\{\},1}+\vartheta_{\{1\},2}}$, then $\lim_{t\to\infty} |i_{v,\{1\}}(t) - i_{v,\{2\}}(t)| = 0$ for all $v \in V$.*

Proof. From the simplified master equation (4.2), we have

$$\begin{aligned} \frac{d}{dt}i_{v,\{1\}}(t) - \frac{d}{dt}i_{v,\{2\}}(t) = &\left[\left(1 - \prod_{(u,v)\in E}(1 - \vartheta_{\{\},1}i_{u,\{1\}}(t))\right)\right. \\ &\left. - \left(1 - \prod_{(u,v)\in E}(1 - \vartheta_{\{\},2}i_{u,\{2\}}(t))\right)\right] \\ &i_{v,\{\}}(t) - \beta_{\{1\},1}(i_{v,\{1\}}(t) - i_{v,\{2\}}(t)) - \\ &\left[\left(1 - \prod_{(u,v)\in E}(1 - \vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)i_{v,\{1\}}(t)\right. \\ &\left. - \left(1 - \prod_{(u,v)\in E}(1 - \vartheta_{\{1\},2}i_{u,\{1\}}(t))\right)i_{v,\{2\}}(t)\right]. \end{aligned}$$

Noting

$$\begin{aligned}
&\left|\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{\},1}i_{u,\{1\}}(t))\right)\right.\\
&\quad\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{\},2}i_{u,\{2\}}(t))\right)\right|\\
&\le\sum_{u\in V}a_{vu}|i_{u,\{1\}}(t)-i_{u,\{2\}}(t)|\vartheta_{\{\},1},
\end{aligned}$$

and

$$\begin{aligned}
&-\left[\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)i_{v,\{1\}}(t)\right.\\
&\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{1\}}(t))\right)i_{v,\{2\}}(t)\right]\\
&\le-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)(i_{v,\{1\}}(t)-i_{v,\{2\}}(t))\\
&+\left|\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)\right.\\
&\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{1\}}(t))\right)\right|i_{v,\{2\}}(t)\\
&\le-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)(i_{v,\{1\}}(t)-i_{v,\{2\}}(t))\\
&+\sum_{u\in V}a_{vu}|i_{u,\{1\}}(t)-i_{u,\{2\}}(t)|\vartheta_{\{1\},2},
\end{aligned}$$

we have

$$\begin{aligned}
\frac{d}{dt}|i_{v,\{1\}}(t)-i_{v,\{2\}}(t)|&\le-\beta_{\{\},1}|i_{v,\{1\}}(t)-i_{v,\{2\}}(t)|\\
&+\sum_{u\in V}a_{vu}|i_{u,\{1\}}(t)-i_{u,\{2\}}(t)|(\vartheta_{\{\},1}+\vartheta_{\{1\},2}).
\end{aligned}$$

Consider a comparison system

$$\frac{d}{dt}z_v(t)=-\beta_{\{\},1}z_v(t)+\sum_{u\in V}a_{vu}z_u(t)(\vartheta_{\{\},1}+\vartheta_{\{1\},2}).$$

One can see that $z_v(t)\ge|i_{v,\{1\}}(t)-i_{v,\{2\}}(t)|$ holds for all $t\ge0$ and $v\in N$ as long as $z_v(0)\ge|i_{v,\{1\}}(0)-i_{v,\{2\}}(0)|$. The given condition implies that the real parts of all eigenvalues of $-\beta_{\{\},1}I_n+(\vartheta_{\{1\},2}+\vartheta_{\{\},1})A$ are negative. According to Lemma 2, $\lim_{t\to\infty}z_v(t)=0$ for all $v\in V$. This implies $\lim_{t\to\infty}|i_{v,\{1\}}(t)-i_{v,\{2\}}(t)|=0$ for all $v\in V$. The proof is completed. □

Theorem 24 (outcome of viruses being approximately equally powerful). *Suppose there is* $\epsilon>0$ *such that* $|\vartheta_{\{\},1}-\vartheta_{\{\},2}|\le\epsilon$, $|\vartheta_{\{1\},2}-\vartheta_{\{2\},1}|\le\epsilon$, $|\beta_{\{1\},1}-\beta_{\{2\},2}|\le\epsilon$, *and* $|\beta_{\{1,2\},1}-\beta_{\{1,2\},2}|\le\epsilon$. *If* $\lambda_1<\frac{\beta_{\{1\},1}}{\vartheta_{\{\},1}+\vartheta_{\{1\},2}}$, *then* $\overline{\lim}_{t\to\infty}|i_{v,\{1\}}(t)-i_{v,\{2\}}(t)|\le d'\epsilon$ *for some* $d'>0$ *independent of* ϵ *and all* $v\in V$.

Proof. From the simplified master equation (4.2), we have

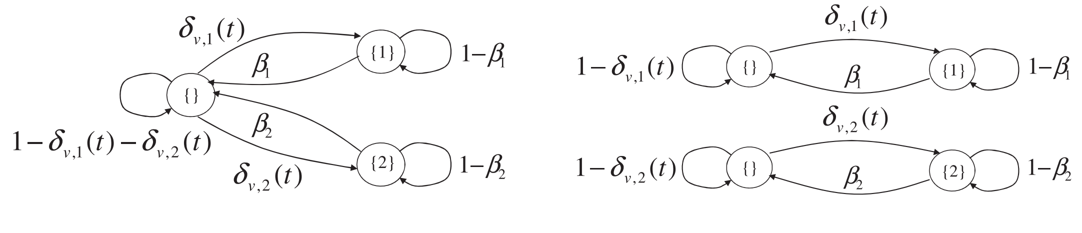
$$\begin{aligned}
&\frac{d}{dt}i_{v,\{1\}}(t)-\frac{d}{dt}i_{v,\{2\}}(t)\\
&=\left[\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{\},1}i_{u,\{1\}}(t))\right)\right.\\
&\quad\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{\},2}i_{u,\{2\}}(t))\right)\right]i_{v,\{\}}(t)\\
&\quad-\beta_{\{1\},1}(i_{v,\{1\}}(t)-i_{v,\{2\}}(t))-(\beta_{\{1\},1}-\beta_{\{2\},2})i_{v,\{2\}}(t)\\
&\quad-\left[\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)i_{v,\{1\}}(t)\right.\\
&\quad\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{2\},1}i_{u,\{1\}}(t))\right)i_{v,\{2\}}(t)\right]\\
&\quad+(\beta_{\{1,2\},2}-\beta_{\{1,2\},1})i_{v,\{1,2\}}(t).
\end{aligned}$$

Noting

$$\begin{aligned}
&\left|\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{\},1}i_{u,\{1\}}(t))\right)\right.\\
&\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{\},2}i_{u,\{2\}}(t))\right)\right|\\
&\le\sum_{u\in V}a_{vu}|i_{u,\{1\}}(t)-i_{u,\{2\}}(t)|\vartheta_{\{\},1}\\
&+\left|\prod_{(u,v)\in E}(1-\vartheta_{\{\},1}i_{u,\{2\}}(t))-\prod_{(u,v)\in E}(1-\vartheta_{\{\},2}i_{u,\{2\}}(t))\right|\\
&\le\sum_{u\in V}a_{vu}|i_{u,\{1\}}(t)-i_{u,\{2\}}(t)|\vartheta_{\{\},1}+\sum_{u}a_{vu}i_{u,\{2\}}(t)|\vartheta_{\{\},1}-\vartheta_{\{\},2}|\\
&\le\sum_{u\in V}a_{vu}|i_{u,\{1\}}(t)-i_{u,\{2\}}(t)|\vartheta_{\{\},1}+\bar{d}\epsilon,
\end{aligned}$$

and

$$\begin{aligned}
&-\left[\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)i_{v,\{1\}}(t)\right.\\
&\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{1\}}(t))\right)i_{v,\{2\}}(t)\right]\\
&\le-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)(i_{v,\{1\}}(t)-i_{v,\{2\}}(t))\\
&+\left|\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{2\},1}i_{u,\{2\}}(t))\right)\right.\\
&\left.-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{1\}}(t))\right)\right|i_{v,\{2\}}(t)\\
&\le-\left(1-\prod_{(u,v)\in E}(1-\vartheta_{\{1\},2}i_{u,\{2\}}(t))\right)(i_{v,\{1\}}(t)-i_{v,\{2\}}(t))\\
&+\sum_{u\in V}a_{vu}|i_{u,\{1\}}(t)-i_{u,\{2\}}(t)|\vartheta_{\{1\},2}+\bar{d}\epsilon,
\end{aligned}$$



(a) disjoint spreading

(b) independent spreading

Fig. 5. State transition diagrams of node $v \in V$ with $|\mathcal{J}| = 2$: disjoint spreading versus independent spreading.

we have

$$\frac{d}{dt}|i_{v,\{1\}}(t) - i_{v,\{2\}}(t)| \le -\beta_{\{\},1}|i_{v,\{1\}}(t) - i_{v,\{2\}}(t)| + \sum_{u\in V} a_{vu}|i_{u,\{1\}}(t) - i_{u,\{2\}}(t)|(\vartheta_{\{\},1} + \vartheta_{\{1\},2}) + d'\epsilon,$$

where $d' = 2\bar{d} + 3$. Via Lemma 1, the given condition implies $-\beta_{\{\},1}I_n + (\vartheta_{\{1\},2} + \vartheta_{\{\},1})A$ is an M-matrix. The theorem can be proved in the same fashion as in the proof of Theorem 16. □

The above analytical results allow us to draw:

Insight 25. *If two viruses have (approximately) equal infection power and are (approximately) equally defended against, then they will have (approximately) the same fate, no matter how many nodes they initially infect and no matter they will die out or not. Moreover, if they die out, they will die out (asymptotically) at the same time.*

5 Further Insights Offered by the Theoretical Results

First, by comparing Corollary 5 and Corollary 7, we draw:

Insight 26. *The (reactive) countermeasures, which are sufficient for killing the spreading of multiple viruses when some (but not many) nodes are initially infected, might not be sufficient for killing the spreading of multiple viruses when many nodes are initially infected. Therefore, preventive/proactive defense and rapid attack detection/response are always very valuable.*

Second, consider a special case of the model investigated in Section 3, namely that the viruses spread disjointly *but* without being able to rob each other, meaning $\vartheta_{\ell,j} = 0$ for $\ell, j \in \mathcal{J}$. Then, the state transition diagram in Fig. 1 is now simplified as Fig. 5a, and the sufficient condition given in Corollary 9 becomes $\lambda_1 < \min_{j\in\mathcal{J}}\{\frac{\beta_j}{\gamma_j}\}$. One the other hand, recall the sufficient condition (due to [10]) for the dying out of a *single* virus spreading in an arbitrary network is $\lambda_1 < \frac{\beta}{\gamma}$. As a trivial extension, this means that when multiple viruses spread *independently* over the same network, the sufficient condition for virus j to die out is $\lambda_1 < \frac{\beta_j}{\gamma_j}$. For the case of two viruses, the state transition diagram is Fig. 5b. Also considering Corollaries 10 and 18, the above discussion leads to:

Insight 27. *The (reactive) countermeasures, which are sufficient for killing the independent spreading of viruses when* many *nodes are initially infected, are 1) sufficient for killing the* disjoint *spreading of viruses when* some *nodes are initially infected, 2) probably not sufficient for killing the* disjoint *spreading of viruses when* many *nodes are initially infected, and 3) probably not sufficient for killing the* nondisjoint *spreading of viruses under certain circumstances (e.g., $\beta_{\{1\},1} - \beta_{\{1,2\},2} = 0$ or $\beta_{\{2\},2} - \beta_{\{1,2\},1} = 0$ when there are two viruses).*

The above 1) and 2) characterize the power disjoint versus independent spreading in terms of the tolerable number of initially infected nodes. It may seem counterintuitive, but is actually reasonable because the sufficient condition deals with the dying out of viruses, and nothing else (e.g., what happens when the viruses do not die out). Note that there is no contradiction between this statement and Insight 14, where we claimed that the stronger viruses could be exploited to *alleviate* (but not kill) the weaker viruses that are actually harder for the defender to kill directly.

Third, by considering Theorems 11 and 12, and further Corollaries 9 and 10, we draw:

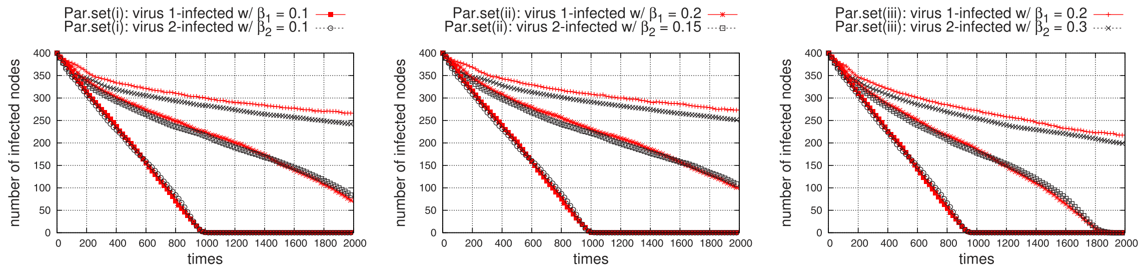
Insight 28. *When some viruses can be robbed by, but cannot rob, the others, it is easier to kill these weak viruses when they initially infected some nodes than when they initially infected many nodes. Moreover, weak viruses may be killed without strong viruses being killed.*

The above insight cannot be directly applied to the case where the viruses spread nondisjointly and do not rob each other. Still, by comparing Theorems 20 and 21, we draw:

Insight 29. *When the viruses do not rob each other and spread nondisjointly (but not independently), it is easier to kill a subset of viruses when they initially infected some nodes than when they initially infected many nodes.*

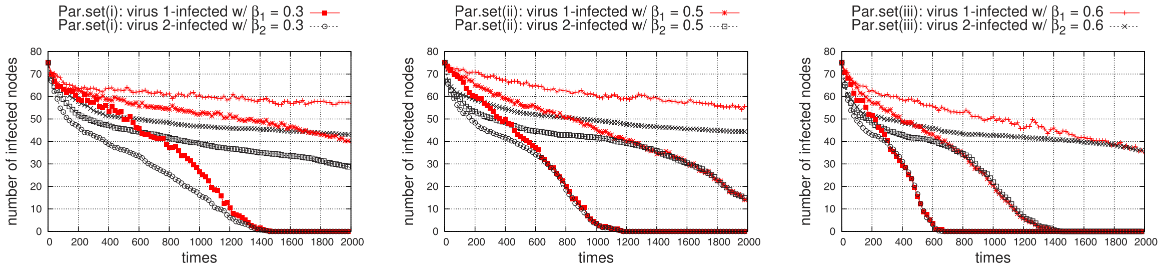
6 Simulation Study

We use simulation (under the same model assumption) to confirm the analytical results in the disjoint spreading model (Section 3) because of its simplicity. We consider three examples of $G = (V, E)$: Erdos-Renyi random graph, regular graph, and power law graph. In each case, we have $|V| = 2{,}000$. The random graph has average node degree 6.001 and $\lambda_1 = 7.1424$. The regular graph has node degree 6 and $\lambda_1 = 6$. The power law graph, which is generated using Brite [23], has average node degree 5.994 and $\lambda_1 = 11.6514$. The graphs have approximately the same average degree because we want to observe, as a side product, whether there is any difference caused by topology. We simulated two settings with $|\mathcal{J}| = 2$ and $|\mathcal{J}| = 3$; in what follows we only report the results corresponding to $|\mathcal{J}| = 2$ because of space limitation ($|\mathcal{J}| = 3$ exhibits similar outcomes though).



(a) random graph (b) regular graph (c) powerlaw graph

Fig. 6. Simulation confirmation of Corollary 9 with two viruses.



(a) random graph (b) regular graph (c) powerlaw graph

Fig. 7. Simulation confirmation of Corollary 10 with two viruses.

Other parameters will be specified below. To represent the dynamics, we plot the number of nodes infected by virus $j \in \mathcal{J}$ at time t based on the average of 20 independent simulation runs.

6.1 Simulation Study for Verifying the Sufficient Conditions for the Dying Out of Viruses

Simulation confirmation of Corollary 9. We let each virus initially infect 400 or 20 percent randomly selected nodes. For each topology, we use the following three parameter set scenarios: 1) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.1, 0.0005, 0.001)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.1,\ 0.00045,\ 0.002)$; 2) $(\beta_1,\ \gamma_1,\ \vartheta_{1,2}) = (0.2,\ 0.0025,\ 0.001)$ and $(\beta_2,\ \gamma_2,\ \vartheta_{2,1}) = (0.15,\ 0.002,\ 0.002)$; 3) $(\beta_1,\ \gamma_1,\ \vartheta_{1,2}) = (0.2, 0.005, 0.001)$ and $(\beta_2,\ \gamma_2,\ \vartheta_{2,1}) = (0.3,\ 0.0045,\ 0.005)$. In each parameter scenario, the sufficient condition stated in Corollary 9 is satisfied, and thus the analytical result says that the two viruses will die out regardless of the degree of initial infection.

Fig. 6 plots (partially) the simulation results, and shows that the three graphs demonstrate similar phenomenon. For example, both viruses spreading over the regular graph die out at about the 1,000th/4,000th/12,000th step in parameter scenarios 1)/2)/and 3), respectively.

Simulation confirmation of Corollary 10. We let each virus initially infect 75 or 3.75 percent randomly selected nodes. For each topology, we consider the following three parameter set scenarios: 1) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.3, 0.021, 0.1)$ and $(\beta_2,\ \gamma_2,\ \vartheta_{2,1}) = (0.3,\ 0.024,\ 0.2)$; 2) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.5,\ 0.031,\ 0.1)$ and $(\beta_2,\ \gamma_2,\ \vartheta_{2,1}) = (0.5,\ 0.034,\ 0.2)$; 3) $(\beta_1,\ \gamma_1,\ \vartheta_{1,2}) = (0.6, 0.024, 0.1)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.6, 0.048, 0.2)$. In each parameter scenario, the sufficient condition in Corollary 10 is satisfied, and thus the viruses should die out.

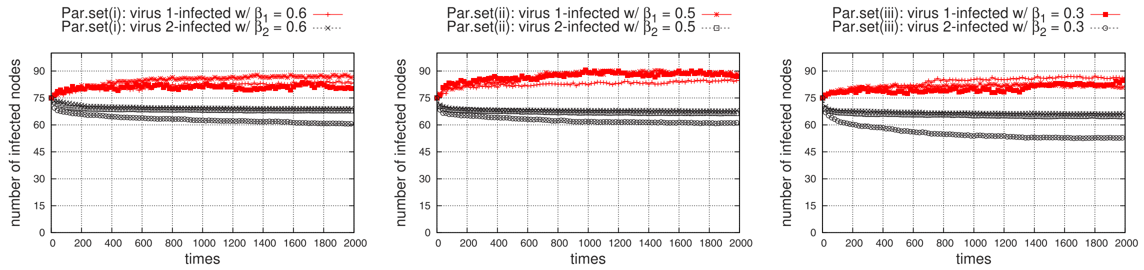
Fig. 7 plots (partially) the simulation results, which demonstrate that the three graphs demonstrate similar phenomenon. For example, Fig. 7b plots the dynamics in the regular graph and shows that both viruses die out at about the 1,200th/2,800th/10,000th step in the parameter scenarios 1)/2)/3), respectively. Note that there is a period of time at the beginning stage, during which the two viruses exhibit different characteristics. This is mainly caused by the fact that $\vartheta_{1,2} = 0.1 < \vartheta_{2,1} = 0.2$, meaning that virus 1's robbing capability is stronger than virus 2's.

Because Corollary 10 hints that its sufficient condition is close-to-being-necessary, we would like to confirm this property as well. We let each virus initially infect 75 or 3.75 percent randomly selected nodes, respectively. For each topology, we consider three parameter scenarios: 1) $(\beta_1,\ \gamma_1,\ \vartheta_{1,2}) = (0.6,\ 0.5,\ 0.1)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.6, 0.048, 0.2)$; 2) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.5, 0.3, 0.1)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.5, 0.034, 0.2)$; 3) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.3, 0.1, 0.1)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = \ (0.3, 0.024, 0.1)$. In every parameter set scenario of every case, the sufficient condition is slightly violated and the viruses may not die out. Fig. 8 plots (partially) the simulation results, which show that in every scenario of every case, the viruses do not die out.

In summary, our simulation confirms Corollaries 9 and 10. As a side-product, our simulation study suggests that it seems more difficult to kill all the viruses in regular and random networks than in power law networks. It is interesting to prove this analytically.

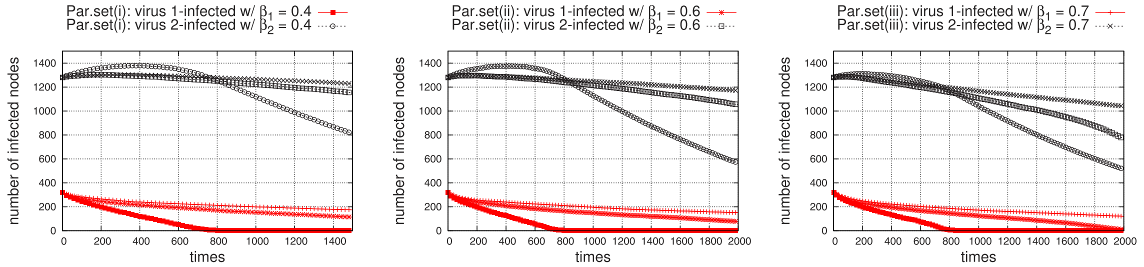
6.2 Simulation Study for Verifying the Outcome when Some Viruses Are More Powerful

Simulation confirmation of Theorem 11. We let virus 1 initially infect 320 or 16 percent randomly selected nodes, and virus 2 initially infect 1,280 or 64 percent randomly selected nodes. Fig. 9 plots the dynamics of the two viruses with respect to the following three parameter scenarios in each case of system graph topologies: 1) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.4, 0.02, 0.2)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.001, 1.0, 0.0)$; 2) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.6, 0.02, 0.3)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.002, 1.0, 0.0)$;

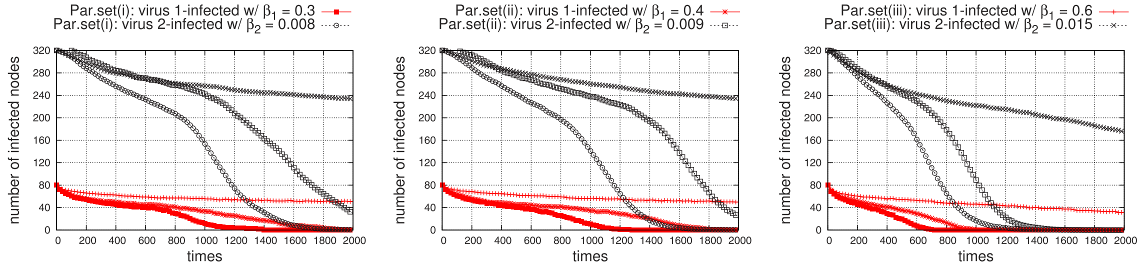


(a) random graph (b) regular graph (c) powerlaw graph

Fig. 8. Simulation confirmation of Corollary 10 with two viruses.



(a) random graph (b) regular graph (c) powerlaw graph



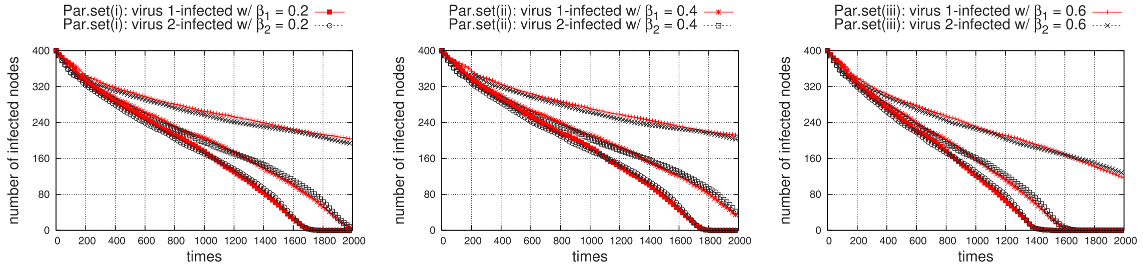
(a) random graph (b) regular graph (c) powerlaw graph

Fig. 10. Simulation confirmation of Theorem 12 with two viruses.

3) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.7, 0.030.4)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.002, 1.0, 0.0)$. Note that in each parameter set scenario, virus 1 is the weaker one (i.e., $\mathcal{J}_1 = \{1\}$) and virus 2 is the stronger one, and the parameters corresponding to virus 1 satisfy the sufficient condition in Theorem 11, which means that virus 1, but not necessarily virus 2, will die out. Note also that we intentionally select the parameters for virus 2 so that the sufficient condition in Corollary 9 is violated; otherwise, virus 2 is bound to die out as well. Fig. 9 plots part of the simulation results. With respect to the parameter scenarios 1)/2)/3), simulation shows that in the random graph, the weaker virus dies out at about the 800th/3,100th/8,000th step, respectively; in the regular graph, the weaker virus dies out at about the 790th/3,100th/8,000th step, respectively; in the power law graph, the weaker virus dies out at about the 800th/2,220th/4,500th step, respectively. Note that in all cases virus 2 dies out much more slowly, which does not contradict with the analytical result.

Simulation confirmation of Theorem 12. We let virus 1 initially infect 80 or 4 percent randomly selected nodes, and virus 2 initially infect 320 or 16 percent randomly selected nodes. Fig. 10 plots (partially) the dynamics of two viruses with the following three parameter scenarios in each case of system graph topologies: 1) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.3, 0.021, 0.1)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.008, 1.0, 0.0)$; 2) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.4, 0.032, 0.4)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.009, 1.0, 0.0)$; 3) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.6, 0.042, 0.4)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.015, 1.0, 0.0)$. In each parameter scenario, virus 1 is the weaker one and virus 2 is the stronger one, and the parameters corresponding to virus 1 satisfy the sufficient condition in Theorem 12, which means that virus 1, but not necessarily virus 2, will die out. With respect to the parameter scenarios 1)/2)/3), simulation shows that in the random graph, the weaker virus dies out at about the 1,500th/2,400th/12,000th step, respectively; in the regular graph, the weaker virus dies out at about the 1,300th/2,300th/11,400th step, respectively; in the power law graph, the weaker virus dies out at about the 800th/1,100th/3,900th step, respectively.

In summary, our simulation study confirms the sufficient conditions given in Theorems 11 and 12, under which the weaker, but not necessarily the stronger, viruses die out.



(a) random graph (b) regular graph (c) powerlaw graph

Fig. 11. Simulation confirmation of Theorem 15 with two viruses.

6.3 Simulation Study for Verifying the Outcome of Viruses Being Equally Powerful

Fig. 11 plots (partially) the dynamics of two viruses with three parameter scenarios: 1) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.2, 0.0018, 0.002)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.2, 0.0018, 0.002)$; 2) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.4, 0.0022, 0.002)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.4, 0.0022, 0.002)$; 3) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.6, 0.0036, 0.002)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.6, 0.0036, 0.002)$. All the parameter scenarios satisfy the sufficient condition in Corollary 9, meaning that the two viruses will die out, and the sufficient condition in Theorem 15, meaning that the two viruses will exhibit the same dynamics. We let each of them initially infect 400 or 20 percent randomly selected nodes. Fig. 11 demonstrates that for each graph in each of the parameter scenario, the two viruses exhibit almost identical dynamics.

Fig. 12 plots (partially) the dynamics of two viruses with the following three parameter scenarios: 1) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.4, 0.005, 0.02)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.4, 0.005, 0.02)$; 2) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.6, 0.0075, 0.03)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.6, 0.0075, 0.03)$; 3) $(\beta_1, \gamma_1, \vartheta_{1,2}) = (0.8, 0.01, 0.04)$ and $(\beta_2, \gamma_2, \vartheta_{2,1}) = (0.8, 0.01, 0.04)$. All of the three parameter sets do not satisfy the sufficient condition in Corollary 9, meaning that the two viruses may not die out, but satisfy the sufficient condition in Theorem 15, meaning that the two viruses will exhibit the same dynamics. We let each of them initially infect 400 or 20 percent randomly selected nodes. Fig. 12 demonstrates that for each graph in each of the parameter scenario, the two viruses exhibit quite identical dynamics. The small, but sometimes noticeable difference is attributed to the fact that the curves are based on the average of 20 runs and the viruses dying out very slow (in which case oscillation is possible).

In summary, out simulation study confirms the analytical result of Theorem 15, namely that if the viruses are equally powerful, then they will eventually exhibit the same dynamics. In particular, if they are to die out, they will die out at the same time.

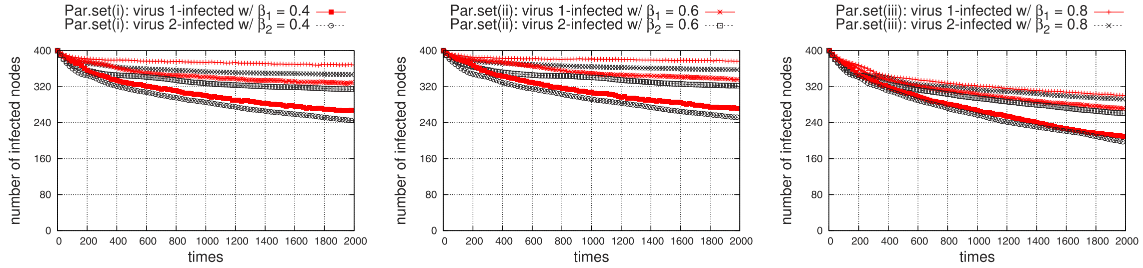
7 Conclusion

We introduced a general model of multivirus spreading dynamics, where the viruses may combat against each other. We presented analytical results in the general model, including sufficient conditions under which the viruses die out when many or some nodes are initially infected, and deeper results for two special cases of the general model. The analytical results made a fundamental connection between the defense capability and the connectivity of the network (via the largest eigenvalue of its adjacency matrix), and allowed us to draw various insights that can be used to guide security defense.

This paper brings a range of new problems for future research. In addition to those mentioned in the body of the paper, we offer: When we state the degree of initial infection, we used the qualitative term "light initial infection," but can we quantify it? What are the necessary conditions under which the viruses will die out? What are the complete dynamics characteristics when (some or all of) the viruses do not die out?

Acknowledgments

We are grateful to the anonymous reviewers for their inspirational comments that helped improve the paper significantly. Shouhuai Xu and Zhenxin Zhan were supported in part by grants sponsored by AFOSR, AFOSR MURI, State of Texas Emerging Technology Fund, and UTSA. The views and conclusions contained in the article are those of the authors and should not be interpreted as, in



(a) random graph (b) regular graph (c) powerlaw graph

Fig. 12. Another simulation confirmation of Theorem 15 with two viruses.

any sense, the official policies or endorsements of the government or the agencies. Wenlian Lu was supported by National Natural Sciences Foundation of China under Grant 60804044, Foundation for the Author of National Excellent Doctoral Dissertation of China under Grant 200921, and Shanghai Rising-Star Program 11QA1400400.

REFERENCES

[1] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," *Proc. the Ninth ACM Conf. Computer and Comm. Security (CCS '02)*, pp. 217-224, 2002.

[2] R. Anderson and R. May, *Infectious Diseases of Humans.* Oxford Univ. Press, 1991.

[3] N. Bailey, *The Mathematical Theory of Infectious Diseases and Its Applications*, second, ed. Hafner Press, 1975.

[4] A. Barabasi and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, pp. 509-512, 1999.

[5] A. Berman and N. Shaked-Monderer, *Completely Positive Matrices.* World Scientific Publishing, 2003.

[6] E. Bursztein, "Extending Anticipation Games with Location, Penalty and Timeline," *Proc. the Fifth Int'l Workshop Formal Aspects in Security and Trust (FAST '08)*, 2008.

[7] E. Bursztein, "NetQi: A Model Checker for Anticipation Game," *Proc. the Sixth Int'l Symp. Automated Technology for Verification and Analysis (ATVA '08)*, pp. 246-251, 2008.

[8] E. Bursztein and J. Goubault-Larrecq, "A Logical Framework for Evaluating Network Resilience against Faults and Attacks," *ASIAN '07: Proc. 12th Asian Computing Science Conf. Advances in Computer Science*, pp. 212-227, 2007.

[9] E. Bursztein and J. Mitchell, "Using Strategy Objectives for Network Security Analysis," *Proc. Fifth Int'l Conf. Information Security and Cryptology (Inscrypt '09)*, 2009.

[10] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos, "Epidemic Thresholds in Real Networks," *ACM Trans. Information and System Security*, vol. 10, no. 4, pp. 1-26, 2008.

[11] R. Chinchani, A. Iyer, H. Ngo, and S. Upadhyaya, "Towards a Theory of Insider Threat Assessment," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '05)*, pp. 108-117, 2005.

[12] M. Dacier and Y. Deswarte, "Privilege Graph: An Extension to the Typed Access Matrix Model," *Proc. Third European Symp. Research in Computer Security (ESORICS '94)*, pp. 319-334, 1994.

[13] A. Ganesh, L. Massoulie, and D. Towsley, "The Effect of Network Topology on the Spread of Epidemics," *Proc. IEEE INFOCOM '05*, 2005.

[14] J. Hale, *Ordinary Differential Equations.* Interscience, 1969.

[15] H. Hethcote, "The Mathematics of Infectious Diseases," *SIAM Rev.*, vol. 42, no. 4, pp. 599-653, 2000.

[16] R. Horn and C. Johnson, *Matrix Analysis.* Cambridge Univ. Press, 1985.

[17] R. Horn and C. Johnson, *Topics in Matrix Analysis.* Cambridge Univ. Press, 1991.

[18] S. Jha and J. Wing, "Survivability Analysis of Networked Systems," *Proc. 23rd Int'l Conf. Software Eng. (ICSE '01)*, pp. 307-317, 2001.

[19] J. Kephart and S. White, "Directed-Graph Epidemiological Models of Computer Viruses," *Proc. IEEE Symp. Security and Privacy*, pp. 343-361, 1991.

[20] J. Kephart and S. White, "Measuring and Modeling Computer Virus Prevalence," *Proc. IEEE Symp. Security and Privacy*, pp. 2-15, 1993.

[21] W. Kermack and A. McKendrick, "A Contribution to the Mathematical Theory of Epidemics," *Proc. Royal Soc. London A*, vol. 115, pp. 700-721, 1927.

[22] A. McKendrick, "Applications of Mathematics to Medical Problems," *Proc. Edinburgh Math. Soc.*, vol. 14, pp. 98-130, 1926.

[23] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An Approach to Universal Topology Generation," *Proc. Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems (MASCOTS '01)*, pp. 346-356, 2001.

[24] V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing, "Ranking Attack Graphs," *Proc. Int'l Symp. Recent Advances in Intrusion Detection (RAID '06)*, pp. 127-144, 2006.

[25] Y. Moreno, R. Pastor-Satorras, and A. Vespignani, "Epidemic Outbreaks in Complex Heterogeneous Networks," *European Physical J. B*, vol. 26, pp. 521-529, 2002.

[26] R. Naraine, "'Friendly' Welchia Worm Wreaking Havoc," http://www.internetnews.com/ent-news/article.php/3065761/Friendly-Welchia-Worm-Wreaking-Havoc.htm, Aug. 2003.

[27] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient Minimum-Cost Network Hardening via Exploit Dependency Graphs," *Proc. 19th Ann. Computer Security Applications Conf. (ACSAC '03)*, pp. 86-95, 2003.

[28] R. Ortalo, Y. Deswarte, and M. Kaaniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security," *IEEE Trans. Software Eng.*, vol. 25, no. 5, pp. 633-650, Sept./Oct. 1999.

[29] R. Pastor-Satorras and A. Vespignani, "Epidemics and Immunization in Scale-Free Networks," *Handbook of Graphs and Networks: From the Genome to the Internet.* Wiley-VCH, 2003.

[30] R. Pastor-Satorras and A. Vespignani, "Epidemic Dynamics and Endemic States in Complex Networks," *Physical Rev. E*, vol. 63, p. 066117, 2001.

[31] R. Pastor-Satorras and A. Vespignani, "Epidemic Dynamics in Finite Size Scale-Free Networks," *Physical Rev. E*, vol. 65, p. 035108, 2002.

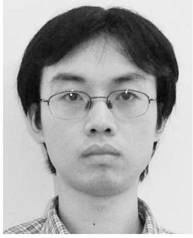
[32] Y. Pesin, "Characteristic Lyapunov Exponents and Smooth Ergodic Theory," *Russ Math Survey*, vol. 32, no. 4, pp. 55-114, 1977.

[33] C. Phillips and L. Swiler, "A Graph-Based System for Network-Vulnerability Analysis," *Proc. 1998 Workshop New Security Paradigms (NSPW '98)*, pp. 71-79, 1998.

[34] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, "Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint," *Proc. 22nd IEEE Symp. Reliable Distributed Systems (SRDS '03)*, pp. 25-34, 2003.

[35] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien, "An Effective Architecture Algorithm for Detecting Worms with Various Scan," *Proc. Fourth Int'l Workshop Peer-to-Peer Systems (IPTPS '05)*, 2005.

Shouhuai Xu received the PhD degree in computer science from Fudan University, China. He is an associate professor in the Department of Computer Science, University of Texas at San Antonio. His research interests include cryptography and cyber security modeling and analysis. More information about his research can be found at www.cs.utsa.edu/.



Wenlian Lu received the BS degree in mathematics and the PhD degree in applied mathematics from Fudan University in 2000 and 2005, respectively. He is an associate professor in the School of Mathematical Sciences and a senior member of the Center for Computational Systems Biology, Fudan University, China. He was a postdoctoral researcher in the Max Planck Institute for Mathematics, Leipzig, Germany, from 2005 to 2007. His current research interests include neural networks, nonlinear dynamical systems, and complex systems.

Zhenxin Zhan received the MS degree in computer science from the Huazhong University of Science and Technology, China, in 2008. He is currently working toward the PhD degree in the Department of Computer Science, University of Texas at San Antonio. His primary research interests are in cyber attack detection and malware analysis.

Long-Range Dependence as Exhibited by Honeypot-Captured Cyber Attacks: The Phenomenon, Implication and Mysterious Cause

***Abstract*—In this paper we report the Long-Range Dependence (LRD) phenomenon that is exhibited by honeypot-captured cyber attacks. To our knowledge, LRD was not known to be relevant in the cyber security domain until now, although it has been known for two decades that LRD is exhibited by benign traffic (i.e., no attacks). We demonstrate the cyber security implications of the LRD phenomenon, by showing how to predict the rates of incoming attacks with reasonable good precision, especially at the scale of "aggregation" the victims as a network. This implies the possibility of providing real-life defenders with sufficient early-warning time for adjusting their defense configurations (e.g., proactively allocating more resources for deep packet inspection). We also explore the cause of the LRD phenomenon. We find that the theory, which was developed for explaining the possible cause of LRD in benign traffic, is probably not applicable to the LRD observed in the cyber security domain. We further find that LRD observed in the cyber security domain is unlikely caused by the intensity of attacks that are launched by individual attackers.**

***Keywords*-Cyber security, long-range dependence (LRD), attack prediction, stochastic cyber attack processes, attack inter-arrival time, attack rate**

I. Introduction

This paper studies statistical phenomena/properties of cyber attacks as exhibited by honeypot-captured attack data. Studies of this kind are important because they can lead to deeper understanding of cyber security from a perspective that is different from most existing studies. For example, the statistical properties may have important real-life implications that can be exploited for more effective cyber defense. Such insights often cannot be obtained from studies at some lower levels of abstractions.

Our Contributions

In this paper, we make two contributions. First, we introduce the concept of *stochastic cyber attack processes*, which are a new kind of mathematical objects for modeling certain cyber attacks. We then present a formal statistical methodology for analyzing them. The methodology is centered on answering three questions: (1) What phenomena/properties do the processes exhibit (e.g., are they Poisson)? (2) What are the implications of the phenomena/properties (e.g., can we exploit them for better prediction of the incoming attacks)? (3) What are the cause of the observed phenomena? Although the present paper only deals with the stochastic cyber attack processes that correspond to honeypot-captured cyber attacks, the methodology can be applied to analyze other kinds of cyber attack data, such as the attacks against (individual computers in) production network — if such data is, fortunate enough, available.

Second, our formal statistical analysis of honeypot-captured data (corresponding to 166 IP addresses for five periods of time, or 220 days accumulatively) led to the following findings. (1) The stochastic cyber attack processes are not Poisson but majority of the processes instead exhibit Long-Range Dependence (LRD), a phenomenon that was not known to be relevant in the cyber security domain until now. (2) We explore how to exploit the presence of LRD to conduct more accurate predictions for the attack processes corresponding to individual victims, and especially for the attack processes corresponding to the victim network (i.e., by aggregating the relevant victims as a whole). This shows the power of "gray-box" (rather than "black-box") prediction because the prediction models can accommodate the LRD phenomenon. (3) We explored the cause of LRD as exhibited by the honeypot-captured data. We confirm that non-stationarity of processes can cause "illegitimate" LRD. However, for the "legitimate" LRD (i.e., not caused by non-stationarity of processes), we find that LRD is probably not caused by that the underlying stochastic attack (sub-)processes exhibit the heavy-tail phenomenon; this is in contrast to the setting of benign traffic, where LRD was believed to be caused by the superposition of processes that exhibit the heavy-tail phenomenon [1]. We further find that the LRD observed here was probably not caused by intense (consecutive) attacks that are launched by individual attackers. This means that more studies are needed before we can actually pin down the actual fundamental cause of LRD in cyber security domain.

Related Work

The present study is based on honeypot-collected cyber attack data. Previous studies of such data are mainly from other perspectives, such as analyzing the honeypot-observed probing activities [2], characterizing/grouping attacks [3], [4], [5], [6], [7], [8], [9], [10], [11], and identifying methods to detect attacks such as DoS (denial-of-service) [12], scans [13], worms [14], [15], and botnets [16], [17]. In contrast, we study the statistical properties of the attack processes corresponding to honeypot-emulated services (or ports), and the feasibility of exploiting the identified properties for predicting the incoming attacks.

LRD was observed in benign traffic about two decades ago [18], [19], [20], [21]. To our knowledge, LRD is identified here to be relevant in the cyber security domain.

The rest of the paper is organized as follows. Section II briefly review some statistical preliminaries. Section III introduces the data we analyze. Section IV describes the identification of LRD that is exhibited by the attack processes corresponding to individual victim honeypot IP addresses. Sections V and VI explore, at the scale of victims (i.e., each attack process corresponds to a victim), the exploitation of the LRD for predicting incoming attacks and the cause of LRD. Section VII studies the same data but from a more macroscopic perspective, by treating the entire honeypot as a victim network. Section VIII discusses the limitations of the present study. Section IX concludes the paper with future research directions.

II. Statistical Preliminaries

Long-Range Dependence (LRD)

The LRD phenomenon has received much attention in contexts other than cyber security [22], [23], [24], [21]. A stationary time sequence $\{X_i, i \geq 1\}$ is said to possess LRD if its autocorrelation function

$$\rho(h) = \mathrm{Cor}(X_i, X_{i+h}) \sim h^{-\beta} L(h), \quad h \to \infty, \tag{1}$$

for $0 < \beta < 1$, and $L(\cdot)$ is a slowly varying function (cf. [25]). That is, for all $t > 0$,

$$\lim_{x \to \infty} \frac{L(tx)}{L(x)} = 1.$$

Note that $\sum_h \rho(h) = \infty$, which captures the intuition behind LRD that while high-lag autocorrelations are small but their cumulative effect is important. The other definition of LRD is based on spectral density [26], [27]. A stationary time series is said to have LRD if its spectral density at frequency λ is proportional to $\lambda^{-2|d|}$, $d \neq 0$ as $\lambda \to 0$. These definitions are equivalent.

The statistical feature of LRD time series is that the degree of LRD can be expressed using one parameter, which is called Hurst parameter (H). The parameter β in Eq. (1) is related to the Hurst parameter via the equation $\beta = 2 - 2H$. Therefore, for LRD, we have $1/2 < H < 1$, and the degree of LRD increases as $H \to 1$. The parameter in spectral density is related to Hurst parameter via the equation $H = .5 + |d|$. In the Appendix, we briefly review six popular methods for estimating the Hurst parameter, which are used in our analysis.

Methods for removing illegitimate LRD

It is known that LRD can also be explained by the processes that are not stationary long-range dependent [28]. In the literature, the processes that have been found to generate illegitimate long memory which include: a short-range dependent time series with change points in the mean; slowly varying trends with random noise; a stationary parametric time series model with time-varying parameters, etc [27], [29].

Therefore, we need to remove the attack processes that exhibit illegitimate LRD. The following method is for this purpose, which tests for the null hypothesis that a given time series is a stationary LRD process against the alternative hypothesis that it is affected by change points or a smoothly varying trend [27]. The test is in the frequency domain and is based on the derivatives of the profiled local Whittle likelihood function in a degenerating neighborhood of the origin. More specifically, we are testing for:

$$H_0\colon X_t \text{ is stationary with LRD.}$$

VS

$$H_a\colon X_t = Z_t + \mu_t \text{ with } \mu_t = \mu_{t-1} + \psi_t \eta_t,$$

where Z_t is a stationary short memory process, η_t is a white noise process and ψ_t is a Bernoulli random variable which takes value 1 with probability p_n. We also test for the alternative

$$H_a\colon X_t = Z_t + h(t/n),$$

where $h(\cdot)$ is a Lipschitz continuous function on $[0, 1]$.

Two statistical models

Throughout this paper, we call a model *LRD-less* if it cannot accommodate LRD, and call a model *LRD-aware* if it can accommodate LRD. The purpose is to show that prediction of cyber attacks should be based on LRD-aware models for better accuracy when the cyber attack data exhibited the LRD phenomenon. Specifically, we consider two popular models of these kinds. Let ϵ_t be independent and identical normal random variables with mean 0 and variance σ_ϵ^2.

- The LRD-less ARMA model: Autoregressive moving average process of orders p and q model, abbreviated to ARMA(p,q), is one of the most popular models in time series [26], with

$$Y_t = \sum_{i=1}^{p} \phi_i Y_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}.$$

- The LRD-aware FARIMA model: A well-known LRD-aware model [22], [23], [24] is the Fractional ARIMA(p, d, q) (FARIMA) with $0 < d < 1/2$ and $H = d + 1/2$. Specifically, define polynomials

$$\phi(x) = 1 - \sum_{j=1}^{p} \phi_j x^j \quad \text{and} \quad \psi(x) = 1 + \sum_{j=1}^{q} \psi_j x^j.$$

  Let X_t be a stationary process such that

$$\phi(B)(1 - B)^d X_t = \psi(B)\epsilon_t,$$

  for some $-1/2 < d < 1/2$, where B is the back shift operator defined by $BX_t = X_{t-1}$, $B^2 X_t = X_{t-2}$, and so on. Then X_t is called FARIMA(p, d, q).

Measurement of prediction accuracy

Suppose $X_m, X_{m+1}, \ldots, X_z$ are observed data (all non-negative), and $Y_m, Y_{m+1}, \ldots, Y_z$ are the predicted data. We can define the prediction error $e_t = X_t - Y_t$, where $m \leq t \leq z$. Recall the popular statistic PMAD (Percent Mean Absolute Deviation):

$$\mathsf{PMAD} = \frac{\sum_{t=m}^{z} |e_t|}{\sum_{t=m}^{z} X_t}.$$

While PMAD can be seen as the overall prediction error, we can define a variant we call *underestimation error*, which counts only the underestimations as follows:

$$\mathsf{PMAD}' = \frac{\sum_{t=m}^{z} e_t}{\sum_{t=m}^{z} X_t} \quad for\ e_t > 0\ and\ corresponding\ x_t.$$

This is relevant because if the defender is willing to over-provision some defense resources, the predicted results can be more useful because it is, as we will see, relatively easier to contain underestimation error (which can correspond to attacks that can be overlooked due to the insufficient provisioning of resources).

We also found it convenient, when dealing with the prediction accuracy at the level/scale of victims, to define prediction *overall accuracy*, or OA for short, based on the PMAD statistic as follows:

$$\mathsf{OA} = 1 - \text{PMAD} = 1 - \frac{\sum_{t=m}^{z} |e_t|}{\sum_{t=m}^{z} X_t}.$$

Moreover, in the context of the present paper, overestimation (i.e., $Y_j > X_j$ for some j's) meaning the over-provisioning of resources in expecting incoming attacks, but underestimation (i.e., $Y_j < X_j$ for any j) can cause the overlook of attacks (i.e., a sort of false-negative because the defender does not allocate enough resource to examine the incoming traffic). Therefore, it makes a good sense to define the following *underestimation accuracy*, or UA for short:

$$\mathsf{UA} = 1 - \frac{\sum_{t=m}^{z} e_t}{\sum_{t=m}^{z} X_t} \quad for\ e_t > 0\ and\ corresponding\ x_t.$$

That is, UA disregards the over-estimations.

III. Data Description

The attack traffic data was collected by a honeypot, which ran four popular low-interaction honeypot software programs: Dionaea [30], Mwcollector [31], Amun [32], and Nepenthes [33]. Each honeypot IP address was assigned to one of these programs and was completely isolated from the other honeypot IP addresses. To save resources, a single honeypot computer was assigned with multiple IP addresses and thus ran multiple honeypot software programs. A dedicated computer was used to collect the raw network traffic as `pcap` files, which are timestamped at the resolution of microsecond. The vulnerable services offered by all four honeypot programs are SMB, NetBIOS, HTTP, MySQL and SSH, each of which is associated to a unique TCP port. This means that each IP address (i.e., honeypot software) opens the ports corresponding to these services. We call these ports *production ports*, and the other ports *non-production ports* (because they are associated to no services). The concrete attacks targeting the production ports can be dependent upon the specific vulnerabilities emulated by the honeypot programs (e.g., the Microsoft Windows Server Service Buffer Overflow MS06040 and Workstation Service Vulnerability MS06070 for the SMB service). However, low-interactive honeypots do not capture sufficient information for precisely characterizing the specific attacks.

Period	Time	Duration (days)	# of victim IPs
I	11/04/2010 - 12/21/2010	47	166
II	02/09/2011 - 02/27/2011	18	166
III	03/12/2011 - 05/06/2011	54	166
IV	05/09/2011 - 05/30/2011	21	166
V	06/22/2011 - 09/12/2011	80	166

Table I: Data description

Table I summarizes the datasets, which correspond to 166 honeypot IP addresses and five periods of time. These periods are not strictly consecutive because of network/system maintenance etc.

A preprocess procedure was used to translate the raw `pcap` data into attack-oriented data format. First, we distinguish the attack traffic based on whether the attacks are against the production ports or against the non-production ports. We focus on the attack traffic against the production ports, which lead to some interactions with the attackers. These interactions led to the attack data that is suitable for the purpose of our analysis.

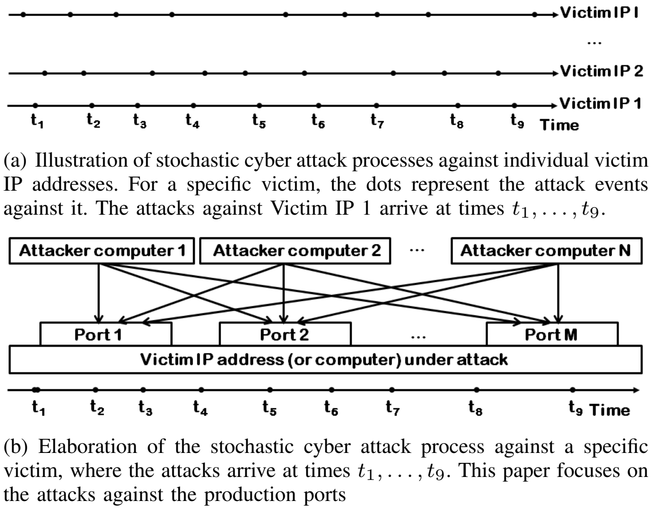
Second, we treat the incoming TCP flows as incoming attacks because the honeypot-emulated vulnerable services are based on TCP. A TCP flow can be extracted from the `pcap` data because it is uniquely identified by the attacker's IP address, the port used by the attacker, the victim IP address in the honeypot, and the port that is under attack. An unfinished TCP handshake is also treated as a flow, and thus an attack. This is reasonable because an unsuccessful handshake can be caused by events such as that the honeypot port in question is busy (i.e., the connection is dropped). For flows that do not end with the FIN flag (indicating safe termination of a TCP connection) or the RST flag (indicating unnatural termination of a TCP connection), we used the following two parameters in the preprocess.

1) Flow timeout time: A flow is considered expired when no packet of the flow is received during a time window of 60 seconds, which is reasonable because honeypot system provides limited interactions.
2) Flow lifetime: A flow is considered expired when a flow lives longer than a pre-determined lifetime, which is set to be 300 seconds also due to the fact that honeypot systems provide limited interactions [34].

IV. The Long-Range Dependence Phenomena

A. Methodology

We found it convenient to analyze honeypot-captured cyber attack data from a *victim* perspective, where a victim corresponds to a single honeypot IP address. This is reasonable because in production networks, an IP address often (albeit not always) represents a computer and reflects a surface that is exposed to the adversary. We also found it convenient to treat the attacks against a victim as a stochastic process, dubbed *stochastic cyber attack process*, which is in principle a kind of Point Process [35]. Figure 1(a) illustrates the attacks against individual victim IP addresses, where the dots on the time line corresponding to a specific victim formulates an attack process. Figure 1(b) further elaborates the idea, where a victim is attacked by N attackers (or attacking computers) and the attacks arrive at time $t_1, \ldots, t_9$. Attacks may target *production ports* that are associated to some services, or *non-production ports* that are not associated to any service. Since attacks targeting the non-production ports can be simply dropped by the computer, we focus on the attacks against the production ports. This is reasonable because real-life attacks often target production ports (one exception is the denial-of-service attacks, which are out of scope of the present paper).



(a) Illustration of stochastic cyber attack processes against individual victim IP addresses. For a specific victim, the dots represent the attack events against it. The attacks against Victim IP 1 arrive at times $t_1, \ldots, t_9$.

(b) Elaboration of the stochastic cyber attack process against a specific victim, where the attacks arrive at times $t_1, \ldots, t_9$. This paper focuses on the attacks against the production ports

Figure 1: Illustration of stochastic cyber attack processes

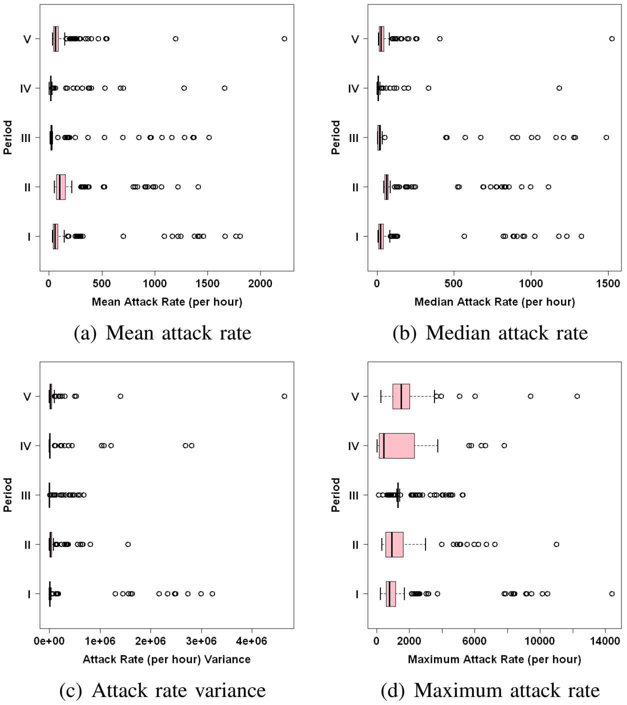
We study the statistical properties of the attack processes from two important perspectives: attack inter-arrival time and attack rate. In either case, we study both basic statistics (which can hint what kinds of advanced statistical analyses are appropriate for further analysis) and the LRD phenomenon they exhibit.

B. Basic Statistics of Attack Rate (Per Hour)

Attack rate describes the number of attacks that arrive at some resolution unit of time (e.g., minute or hour or day). We here consider the *hourly* attack rate, while noting that daily attack rate is not appropriate for the data because the individual datasets correspond to no more than 80 days.

Table II describes, for each period, the observed lower-bound and upper-bound for each statistic among the 166 victims. By taking Period I as an example, we observe the following: there can be no attacks against a specific victim during one hour (as mentioned above, some victim actually was not attacked for up to 7 hours); the average per-hour attack rate (among all the victims) is some number between 32 and 1810 attacks per hour; the median per-hour attack rate is some number between 8 and 1327 attacks per hour; the maximum number of attacks against a single victim can be up to 14403.

Figure 2 presents the boxplots of four statistics: mean, median, maxim, and variance. It clearly shows that the five periods exhibit somewhat similar (homogeneous) statistical properties. For example, there are many outliers in each period and for each statistic.



(a) Mean attack rate

(b) Median attack rate

(c) Attack rate variance

(d) Maximum attack rate

Figure 2: Boxplot of mean attack rate (per-hour). The colored box represents Q_1, which represents the attack rates from the smallest 25%, to Q_3, which represents the largest 25% attack rates. The thick vertical line within the colored box indicates the median attack rate among all the attack rates. An attack rate smaller than $Q_1 - 1.5(Q_3 - Q_1)$ or larger than $Q_3 + 1.5(Q_3 - Q_1)$ is called an outlier. The circles represent outlier attack rates.

Both Table II and Figure 2 hint that the variances are often much larger than the means in all five periods. By looking into all individual attack processes, we found that among all the 166 victims/period × 5 periods = 830 victims, the variance of attack rate is at least 3.5 times greater than the mean attack rate corresponding to the same victim. The fact — the variance is much larger than the mean attack rate — hints that Poisson models may not be appropriate for describing the attack processes. This is confirmed below via formal statistical tests that are presented below.

C. The Cyber Attack Processes Are Not Poisson

It would be ideal that the stochastic cyber attack processes are Poisson because we can easily characterize Poisson processes with very few parameters and there are many mature methods for analyzing them. For example, we can use the property that the superposition of Poisson processes is still a Poisson process [25] to simplify the problems of interest (e.g., when we consider the attacks against an entire network).

Assume that the attack inter-arrival times are independent and identically distributed exponential random variables with distribution

$$F(x) = 1 - e^{-\lambda x}, \ \lambda > 0, x \geq 0.$$

To test the exponential distribution, we first estimate the unknown parameter λ by the maximum likelihood method.

Period (days)	MIN(·)		Mean(·)		Median(·)		Variance(·)		MAX(·)	
	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB
I (47)	0	197	32.053	1810.417	8	1327	1589.907	3219758.756	247	14403
II (18)	0	139	49.843	1411.986	43	1112	1466.538	1553585.624	335	10995
III (54)	0	94	11.514	1513.460	3	1490	254.017	676860.697	125	5287
IV (21)	0	3	3.490	1663.375	1	1184	29.709	2808045.223	41	7793
V (80)	0	35	33.974	2228.840	8.5	1526.5	1225.635	4639659.080	274	12267

Table II: Basic statistics of attack rate (number of attacks per hour). For a specific period and a specific $victim$ (IP address), $\mathsf{MIN}(victim)$ and $\mathsf{MAX}(victim)$ are respectively the smallest and largest per-hour attack rates, $\mathsf{Mean}(victim)$, $\mathsf{Median}(victim)$ and $\mathsf{Variance}(victim)$ are respectively the mean per-hour attack rate, the median per-hour attack rate, and the variance of attack rate. For a specific period and a specific statistic $X \in \{\mathsf{MIN}, \mathsf{Mean}, \mathsf{Median}, \mathsf{Variance}, \mathsf{MAX}\}$, $\forall$ $victim$ and $\forall$ $hour$ we have $X(victim, hour) \geq$ LB and $X(victim, hour) \leq$ UB.

Then, we compute the KS, CM and AD test statistics [36], [37] (cf. Appendix for a review) and compare them against the respective critical values.

Period (days)	KS		CM		AD	
	min	max	min	max	min	max
I (47)	0.127	0.535	482.295	59543.874	inf	inf
II (18)	0.061	0.497	47.082	20437.821	298.728	inf
III (54)	0.060	0.651	163.711	51434.324	1103.704	inf
IV (21)	0.037	0.814	3.438	31376.271	22.832	inf
V (80)	0.083	0.652	323.387	214543.536	inf	inf
CV	0.01215705		0.222		1.132	

Table III: Minimum values of the three test statistics for attack inter-arrival time (unit: second). (Inf) means the value is extremely large.

Table III reports the minimum test statistics, where the critical values for the test statistics are based on significant level .05 and obtained from [38], [39]. Since the values are far from the critical values, there is no evidence to support the exponential distribution hypothesis. Technically, this is because the minimum test statistics violate the exponential distribution assumption, meaning that a greater test statistics must violate the exponential distribution assumption as well.

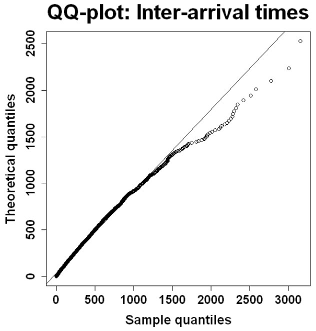


Figure 3: QQ-plot of inter-arrival time corresponding to the specific victim that exhibits the minimum KS, CM and AD value simultaneously (time unit: second)

We also use QQ-plot to evaluate the goodness-of-fit of exponential distributions for the attack inter-arrival time corresponding to the victim that exhibits the minimum test statistics simultaneously in Table III. This is the victim from Period IV with $H_{\mathsf{KS}} = 0.037$, $H_{\mathsf{CM}} = 3.438$ and $H_{\mathsf{AD}} = 22.832$. If the attack inter-arrival time corresponding to this particular victim does not exhibit the exponential distribution, we conclude that no attack inter-arrival time in this dataset exhibits the exponential distribution. The QQ plot is displayed in Figure 3. We observe a large deviation in the tails. Hence, exponential distribution cannot be used as the distribution of attack inter-arrival times, meaning that the attack processes are not Poisson.

D. LRD Exhibited by the Attack Processes

Similarly, Table IV describes the minimums and maximums of the estimated Hurst parameters of attack rates. Consider Period I as an example, we observe that the attack processes corresponding to 163 (out of 166) victims have average Hurst parameters falling into $[.6, 1]$ and thus exhibit LRD, where the average is taken over the six Hurst parameters. However, only the attack processes corresponding to 159 (out of the 163) victims exhibit legitimate LRD, meaning that the LRD corresponding to 4 (out of the 163) victims are actually caused by the non-stationarity of the processes [28]. We also observe that in Period III, there are only 87 victims whose corresponding attack processes exhibit legitimate LRD. Overall, 70% of the stochastic attack processes, or $159 + 116 + 87 + 125 + 89 = 576$ out of $166 \times 5 = 830$ attack processes, exhibit the LRD phenomenon from the perspective of attack rate.

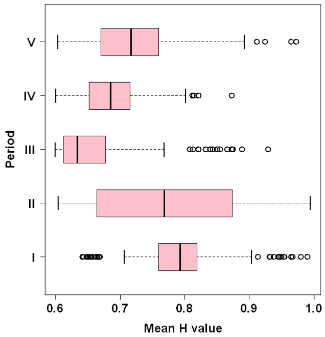


Figure 4: Boxplot of Hurst parameters of attack rate corresponding to individual victims (per hour).

Figure 4 shows the boxplots of Hurst parameters of attack rate. We observe that Periods I and II have relatively large Hurst parameters, suggesting stronger LRD. As we will see in Section V, Period II is somehow more resistant against prediction than Period I. It is also interesting to note that, as we will see in Section VII that Periods I and II can be pretty accurately predicted even for five-hour ahead of time. It remains to be investigated whether this is coincident or causal.

Period	RS		AGV		Peng		Per		Box		Wave		# of victims w/	# of victims w/
(days)	min	max	min	max	min	max	min	max	min	max	min	max	$\bar{H} \in [.6, 1]$	LRD
I (47)	0.533	1.011	0.460	0.977	0.664	1.135	0.725	1.387	0.545	1.153	0.395	0.957	163	159
II (18)	0.494	0.944	0.399	0.979	0.562	1.371	0.529	1.687	0.334	1.321	-0.551	1.330	130	116
III (54)	0.646	0.952	0.299	0.956	0.528	1.059	0.439	1.217	0.430	0.979	0.334	1.015	93	87
IV (21)	0.401	1.133	0.118	1.004	0.494	1.448	0.325	1.738	0.421	1.321	-0.337	1.466	126	125
V (80)	0.519	1.006	0.136	0.985	0.445	1.219	0.474	1.433	0.565	1.304	-0.162	1.175	158	89

Table IV: The estimated Hurst parameters for attack rate (per hour).

E. Summary

We observe 70% of the attack processes exhibit legitimate LRD from the perspective of attack rate. Both the attack inter-arrival time and the attack rate with respect to victim computers or IP addresses exhibit the LRD phenomenon. This means, as in the case of benign traffic [18], [19], [20], that the superposition of attack processes will sustain the burst of attacks, rather than smoothing the burst of attacks. This has three implications. First, defenders should expect that the burst of attacks will sustain. Second, cyber attack processes may only be modeled using LRD-aware stochastic processes. Third, we need to explore how can we exploit the presence of LRD to conduct better prediction for the incoming attacks. This is important because reasonably accurate prediction can give the defenders sufficient early warning time to prepare for the arrival of attacks. In Section V we explore the feasibility of prediction and in Section VI we explore the causes of LRD in this context.

V. Exploiting LRD for Predicting Attack Rate

We ask the following natural question: How many attacks will arrive in the next hours of time? Intuitively, the model that is good at prediction in this context should also accommodate the LRD phenomena, meaning that LRD-aware model might lead to better prediction. Being able to predict the incoming attacks at a reasonable accuracy can lead to insights, which can guide the defense operations of real-life defenders.

A. Prediction Algorithm

Let $\{X_1, \ldots, X_n\}$ be the time series of observed attack rates. The basic idea behind the prediction algorithm is to use some portion of the observed data to build a model (training or model fitting), which is then used to predict the attack rates corresponding to the rest/future portion of the observed data (prediction). In order to have a reliable model, 50% of the observed data is used as the training data for building models. Let h the input parameter, which indicates the step we will predict ahead (i.e., the next time interval during which we predict the number of attacks that will arrive). The algorithm operates as follows (we refer to [40] for more details about time-series prediction methodologies):

1) We divide $\{X_1, \ldots, X_n\}$ into two parts, $\{X_1, \ldots, X_m\}$ and $\{X_{m+1}, \ldots, X_n\}$, where $m = [n/2]$.
2) Repeat the following steps until $m > n - h$.
   a) Fit $\{X_1, \ldots, X_m\}$ to obtain a model denoted by M_m.
   b) Use M_m to predict the number of attacks, denoted by Y_{m+h}, that will arrive during the $(m+h)$th step.
   c) Compute prediction error $e_{m+h} = X_{m+h} - Y_{m+h}$.
   d) Set $m = m + 10$, i.e., adding 10 more observations to the model for refitting purpose in (a).

B. Prediction of Attack Rate

Since there are 166 processes/period × 5 periods = 830 processes prediction results, we have to use Table V to succinctly present the prediction results. The prediction results are with respect to 10-hour ahead predictions during the last 100 hours of each time period. We make the following observations. First, the LRD-aware FARIMA performs reasonably better than the LRD-less ARMA. For example, among the 152 LRD processes in Period I, FARIMA can predict for 29 victims about their 10-hour ahead attack rates with at least 70% overall accuracy, while ARMA can only predict for 13 victims. If the defender is willing to over-provision some resources and mainly cares about the underestimation error (which could cause passing incoming packets without inspecting it while not disrupting the network functions), FARIMA can predict for 40 victims.

Second, the LRD-processes in Period I render more to prediction when compared with the LRD-processes in the other periods.

Third, for non-LRD processes, neither FARIMA nor ARMA can provide good predictions. This may be caused by that (some of) the non-LRD processes are non-stationary. We plan to investigate into this issue in the future.

C. Summary

We found that LRD-aware models *can* predict attack rates of LRD-processes, and that LRD-aware models are relatively more accurate than LRD-less models in prediction with respect to LRD-processes. Moreover, we also found that there are LRD processes that can resist the prediction of even LRD-aware models. This hints that in order to be able to predict incoming attacks hours ahead of time, we need new models.

VI. Exploring Causes of the LRD Phenomenon

We have observed that the attack processes corresponding to 70% (576 out of the 166 victims/period × 5 periods = 830) victims exhibit the legitimate LRD phenomenon (i.e., not caused by the non-stationarity of the processes as discussed in Section IV), and showed how to exploit LRD for better prediction. Now we explore the possible causes of the legitimate LRD exhibited by the data. Despite intensive studies in other settings, the fundamental cause of LRD was mysterious. In the setting of the present paper, we focus on exploring two possible causes.

- In the context of benign traffic, it was also known that LRD is probably caused by the heavy-tail phenomenon

Period	total # of victims $((x_1, x_2)/(y))$	# of victims w/ average OA $\geq 80\%$		# of victims w/ average OA $\geq 70\%$		# of victims w/ average OA $\geq 60\%$		# of victims w/ average UA $\geq 80\%$		# of victims w/ average UA $\geq 70\%$		# of victims w/ average UA $\geq 60\%$	
		FARIMA	ARMA	FARIMA	ARMA	FARIMA	ARMA	FARIMA	ARMA	FARIMA	ARMA	FARIMA	ARMA
I	LRD: (152,152)/(159)	2	1	29	13	81	66	13	4	40	35	89	68
	non-LRD: (7,7)/(7)	0	0	4	4	6	6	1	4	7	6	7	7
II	LRD: (109,109)/(116)	0	0	3	2	9	8	2	1	12	6	26	15
	non-LRD: (50,49)/(50)	0	0	0	0	0	2	4	1	6	2	13	5
III	LRD: (82,82)/(87)	0	0	4	4	8	9	9	5	23	19	50	43
	non-LRD: (79,79)/(79)	0	0	0	0	0	0	0	0	10	7	31	24
IV	LRD: (118,118)/(125)	0	0	2	2	5	6	2	3	4	6	11	14
	non-LRD: (41,39)/(41)	0	0	0	0	0	0	1	0	2	0	4	1
V	LRD: (73,73)/(89)	0	0	0	0	2	1	0	1	2	3	16	4
	non-LRD: (77,61)/(77)	0	0	0	0	1	1	0	0	1	0	24	15

Table V: Number of attack processes that can be pretty actually predicted by the LRD-aware FARIMA model, which is more accurate than the prediction of the LRD-less ARMA model. For the column "total # of victims $((x_1, x_2)/(y))$," y is the total number of victims that exhibited LRD (or non-LRD), x_1 (or x_2) is total number (out of the y) of victims for which the Maximum Likelihood Estimator used in the FARIMA (ARMA) algorithm actually converges (i.e., $y - x_1$ and $y - x_2$ victims cannot be predicted, where the limitation is specific to MLE rather than to our algorithm). The column "# of victims w/ average OA (or UA) $\geq z\%$" represents the average number of victims (out of the x_1 or x_2 victims that can be predicted) that lead to average prediction accuracy in terms of overall-accuracy (or underestimation-accuracy) is at least $z\%$, where average is taken over all prediction steps.

that is exhibited by the underlying sub-processes [1]. In Section VI-A, we investigate whether or not the LRD phenomeon we observed was caused by the heavy-tail phemomenon that is exhibited by the underlying cyber attack sub-processes.

- It is intuitive that the LRD may be caused by that many attackers launch intensive (and consecutive) attacks against individual victims (i.e., the attacks launched by individual attackers are bursty). In Section VI-B, we investigate whether or not this is a probable cause of the LRD.

A. Statistical Properties of Port-centric Processes

In order to investigate whether or not the LRD is caused by that the underlying sub-processes exhibit the heavy-tail phenomenon, we need to decompose an attack process corresponding to a victim into port-centric processes corresponding to the individual ports of the victim. As shown in Figure 5, the attack process corresponding to the victim, which describes the attacks that arrive at time $t_1, \ldots, t_9$, is the superposition of M port-centric sub-processes, where attacks against an individual port are treated as an sub-process.

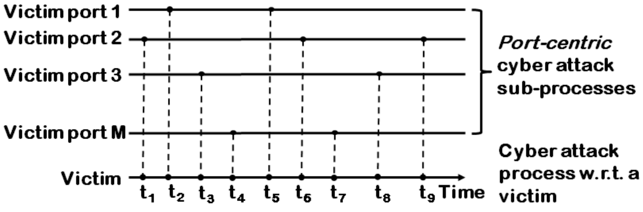


Figure 5: Decomposition of the cyber attack process corresponding to a victim into port-centric sub-processes: the sub-process corresponding to victim Port 1 describes the attacks that arrive at time t_2 and t_5 (the attacks may be launched by one or two attackers); the sub-process corresponding to victim Port 2 describes the attacks that arrive at t_1, t_6 and t_9, etc.

We want to see whether or not the port-centric sub-processes exhibit the heavy-tail phenomenon. Table VI shows that only 8% sub-processes (or $56+80+47+3+32 = 218$ out of the $(159 + 116 + 87 + 125 + 89 = 576)$ victims × 5 ports/victim = 2880 ports) exhibit the heavy-tail phenomenon. Moreover, only 29 (out of the 576) victims have two or three sub-processes that exhibit the heavy-tail phenomenon. Further, there is only 1 port in total whose corresponding sub-processes exhibits infinite mean because the shape value ≥ 1, and there are $1+50+2+1+5 = 59$ ports whose corresponding sub-processes exhibit infinite variance because their shape values $\in (.5, 1)$. The above observations hint that unlike in the setting of benign traffic [1], the LRD phenomenon exhibited by the honeypot-captured attack traffic is probably not caused by the heavy-tail phenomenon exhibited by the underlying attack sub-processes.

B. Statistical Properties of Attacker-centric Processes

In order to figure out whether or not the LRD phenomenon was caused by the intense (consecutive) attacks that are launched by individual attackers, we consider the attacks against each victim that are launched by *distinct* attackers. As illustrated in Figure 6, even though an attacker launched multiple consecutive attacks against a victim, we only consider the first attack. We call the resulting processes *attack-centric*. The purpose of studying attacker-centric processes is the following: If the attacker-centric processes do not exhibit the LRD phenomenon, we conclude that the LRD phenomenon we observe is probably caused by the intensity of the attacks that are launched by individual attackers; otherwise, the LRD is probably not caused by the intensity of the attacks.

Table VII describes the observed lower-bound and upper-bound of the four statistics regarding the attacker-centric processes, where the bounds are among all victims within a period of time. By taking Period II as an example, we observe the following: on average there are between 48 and 100 attackers against one individual victim within one hour, and there can be up to 621 attackers against one individual victim within one hour. Further, attacks in Periods III and IV exhibit different behaviors from the other three periods.

Period	total # of victims exhibiting LRD	# of victims w/ sub-processes exhibiting heavy-tail	# of victims with certain # of sub-processes exhibiting heavy-tail					total # of ports exhibiting heavy-tail	Shape mean value	# of ports w/ shape value $\in (.5, 1)$	# of ports w/ shape value ≥ 1	Standard deviation
			1	2	3	4	5					
I	159	56	50	6	0	0	0	62	.1099567	1	0	.1072166
II	116	80	78	11	1	0	0	103	.4001687	50	0	.224729
III	87	47	39	6	2	0	0	57	.2180889	2	0	.1761706
IV	125	3	3	0	0	0	0	3	.4303879	1	0	.3454725
V	89	32	29	1	2	0	0	37	.2996517	5	1	.2483402

Table VI: Out of the attack processes exhibit the LRD phenomenon, some port-centric sub-processes exhibit the heavy-tail phenomenon.

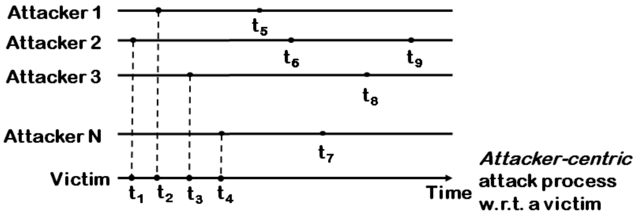


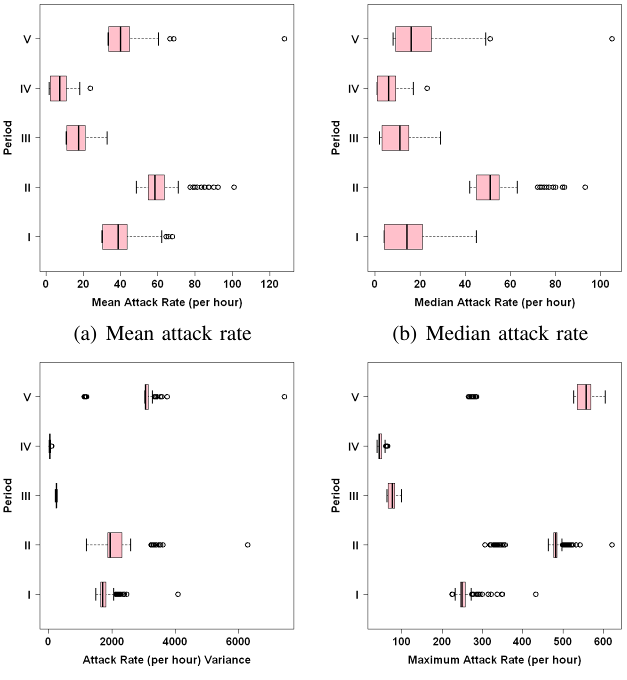
Figure 6: Attacker-centric attack process after ignoring the subsequent attacks launched by the same attacker. For example, we ignore the attack launched by Attacker 1 at time t_5 and the attack launched by Attacker 2 at times t_6 and t_9, etc. That is, the attacker-centric attack process corresponding to the victim describes the attacks that arrive at times t_1, t_2, t_3, t_4.

Period (days)	Mean$(\cdot)$		Median$(\cdot)$		Variance$(\cdot)$		MAX$(\cdot)$	
	LB	UB	LB	UB	LB	UB	LB	UB
I (47)	30.217	67.752	4	45	1498.064	4094.264	225	432
II (18)	48.546	100.774	42	93	1195.136	6298.324	306	621
III (54)	11.059	32.952	2	29	223.638	270.800	64	100
IV (21)	1.938	23.798	1	23	26.322	92.725	40	65
V (80)	33.434	127.944	8	105	1132.646	7465.228	266	605

Table VII: Basic statistics of attack rate of the attacker-centric processes (per hour).

Figure 7 uses boxplot to expose more information (beyond the lower-bounds and upper-bounds mentioned above) about the attacker-centric attack processes in terms of the same statistics. It further shows that the attackers' behaviors are actually very different in all five periods. In particular, it shows that Period II has many outliers in all four statistics, meaning that the attack rate during this period varies a lot.

In order to see whether the attacker-centric processes still exhibit the LRD phenomenon, we describe their Hurst parameters in Table VIII. Using Period I as an example, we observe that there are 153 (out of the 166) victims whose corresponding attacker-centric processes exhibit the LRD phenomenon because their average Hurst parameter $\in [.6, 1]$, where the average is taken over the six Hurst parameter methods. Moreover, none of the 153 attacker-centric processes exhibit the illegitimate LRD phenomenon caused by non-stationarity of the processes. Using Period V as another example, we observe that all 166 victim whose attacker-centric processes have average Hurst parameter $\in [.6, 1]$. However, there are only 77 victims whose attacker-centric exhibit the LRD phenomenon, and the other 89 processes exhibit the illegitimate LRD phenomenon (possibly caused by non-stationarity of processes). The above discussion suggests that the LRD phenomenon exhibited by the attack processes is probably not caused by the intense (consecutive) attacks of the individual attackers.



(a) Mean attack rate

(b) Median attack rate

(c) Attack rate variance

(d) Maximum attack rate

Figure 7: Boxplot of the four statistics of attack rate of the attacker-centric processes (per hour).

C. Summary

First, we have confirmed that LRD indeed can be caused by the non-stationarity of the processes. Second, we have found that, in the cyber security domain as exhibited by the honeypot-captured data we analyzed, LRD is probably not caused by that the underlying port-centric attack sub-processes exhibit the heavy-tail phenomenon. This is in contrast to the domain of benign traffic, where is it believed that LRD was probably caused by the superposition of the underlying processes that exhibit the heavy-tail phenomenon [1]. Third, the LRD exhibited by the attack processes is probably not caused by the intensive (consecutive) attacks that are launched by the individual attackers.

As such, beyond that non-stationarity of processes can cause the LRD phenomenon in the cyber security domain, the fundamental of LRD in stationary processes remain mysterious. This calls for more future studies, which could be rewarded with even better models for predicting incoming cyber attacks.

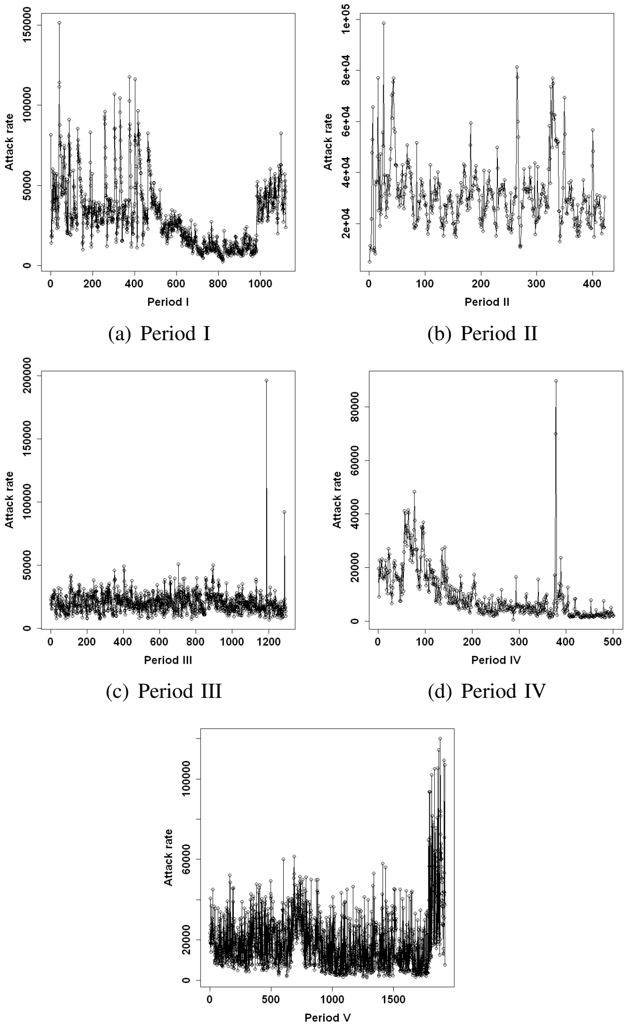
Period	RS		AGV		Peng		Per		Box		Wave		# of victims w/	# of victims w/
(day)	min	max	min	max	min	max	min	max	min	max	min	max	$\bar{H} \in [.6, 1]$	LRD
I (47)	0.593	0.977	0.851	0.958	0.896	1.111	1.174	1.334	0.942	1.185	0.582	0.843	153	153
II (18)	0.570	0.883	0.616	0.950	0.689	1.070	0.710	1.152	0.663	1.242	-0.360	0.728	92	77
III (54)	0.776	0.994	0.364	0.747	0.630	0.748	0.460	0.679	0.608	0.746	0.389	0.668	163	103
IV (21)	0.657	0.920	0.273	0.955	0.690	0.872	0.559	1.206	0.612	0.952	0.288	1.004	166	165
V (80)	0.495	0.758	0.563	0.727	0.499	0.806	0.898	1.114	0.660	0.977	0.567	0.931	166	77

Table VIII: The estimated Hurst parameters of the attack rate of attacker-centric processes (per hour). The six estimation methods are reviewed in the Appendix. Note that a Hurst value being negative or being greater than 1 means that either the estimation method is not suitable or the data (process) is non-stationary. The column "# of victims w/ $\bar{H} \in [.6, 1]$" represents the total number of attacker-centric processes whose average Hurst parameters (where average is among the six kinds of Hurst parameters) $\in [.6, 1]$, which indicates the presence of the LRD phenomenon. However, some LRD phenomenon may be caused by the non-stationarity of the processes and should be eliminated. This is shown in the column "# of victims w/ LRD," which indicates the total number of attacker-centric processes exhibiting the LRD phenomenon that is not caused by the non-stationarity of the processes.

VII. LRD Exhibited at a More Macroscopic Level

In the above sections, we investigated how we identified the LRD phenomenon as exhibited by the attack processes corresponding to *individual* victims, how we could explore LRD for better prediction, and how we exclude some possible causes of the LRD. In this section, we use the same statistical methodology to study the statistical properties of the same attack data, but from a *more macroscopic* perspective, namely by treating the 166 victim honeypot IP addresses as a network. This perspective is equally, if not more, relevant to real-life defense operations because defenders often look attacks from an entire-network perspective.

At this more macroscopic perspective, we can easily expose things because of the aggregate effect. For example, it is feasible now to plot the time series of the attack rate (per hour), namely the total number of attacks against the entire 166 victim space. Figure 8 plots the time series of attacks in the five periods of time. We make the following observations. First, there are some extremely intense attacks during some hours some Periods III and IV. The specific hour corresponding to the extreme value in Period III is Apr 01, 2011, 12 Noon (US Eastern Time); the attacks are against the SSH services. It is evident that the attacks are brute-forcing password. The peak of attacks during Period IV occurred at May 16, 2011, 3 AM (US Eastern time). The intense attacks were against the HTTP service. We tried to find from the Internet whether or not the peaks correspond to (for example) the outbreak of worm or botnet, but no decisive conclusion can be drawn. Second, the five periods exhibit different attack patterns. For example, Periods I, II and V are relatively stationary. Although the five plots exhibit some change-points, a formal statistical analysis (using the method for removing illegitimate LRD, which was reviewed in Section II) shows that there are some change-points in Period III, which correspond to the largest attack rates.



(a) Period I

(b) Period II

(c) Period III

(d) Period IV

(e) Period V

Figure 8: Time series plots of the five periods. The five periods exhibit different patterns. The x-axis is indicates the relative time with respect to the start time for each period.

A. Identifying the LRD Phenomenon

Table IX describes the basic statistics of the attack rate as exhibited by the attack processes corresponding to the victim network during the five periods of time. We observe that on average, the victim network is least intensively attacked during Period IV because the average hourly attack rate is about 9861, which is substantially smaller than the average attack rate during the other periods.

Period (days)	MIN	Mean	Median	Variance	Max
I (47)	2572	30963.180	28263	401243263.217	151189
II (18)	5155	31576.838	29594	167872818.950	98527
III (54)	6732	20382.262	19579	72436071.490	196210
IV (21)	637	9861.132	6528	93209085.323	89718
V (80)	1417	18960.215	15248.5	205276388.408	120221

Table IX: The basic statistics of attack rate corresponding to the victim network during the five periods of time.

Table X describes the six kinds of Hurst parameters corresponding to the five attack super-processes. Although the Hurst parameters suggests that they all exhibit the LRD phenomenon, a further analysis showed the the LRD phenomenon exhibited by Period III is illegitimate, meaning that it is caused by the non-stationarity of the process.

Period	RS	AGV	Peng	Per	Box	Wave	Legitimate LRD?
I	0.795	0.945	0.877	1.031	0.996	0.754	Yes
II	0.737	0.594	0.855	0.746	0.967	0.842	Yes
III	0.740	0.522	0.646	0.626	0.628	0.645	No
IV	1.053	0.971	0.949	1.072	0.967	1.216	Yes
V	0.741	0.781	0.739	1.026	0.797	0.795	Yes

Table X: The estimated Hurst parameters for attack rate corresponding to the victim network during the five periods of time.

B. Exploiting LRD for Predicting Incoming Attacks

Now we report how we can exploit the LRD phenomenon to enhance the accuracy of predicting the rates of attacks against the victim network.

Table XI describes the prediction accuracy when treating the 166 victim IP addresses as a victim network. We observe the following. First, for Period I and II, both one-hour ahead and five-hour ahead FARIMA predictions are pretty accurate, with error no greater than 22%. However, the ten-hour ahead FARIMA prediction is pretty bad. This means that LRD-aware FARIMA can effectively predict the attack rate pretty accurately even five hours ahead. This would give the defender enough early-warning time.

Period	$\mathsf{PMAD}_{\mathrm{FARIMA}}$	$\mathsf{PMAD}_{\mathrm{ARMA}}$	$\mathsf{PMAD}'_{\mathrm{FARIMA}}$	$\mathsf{PMAD}'_{\mathrm{ARMA}}$
One-hour ahead prediction ($p = 0.5$)				
I	0.1790597	0.4457219	0.1727588	0.1570264
II	0.2171558	0.3629267	0.1488566	0.1493358
III	0.2981511	0.2729375	0.3048926	0.3119207
IV	0.5483212	0.5256603	0.1259078	0.1065005
V	0.5165464	0.5285076	0.4238902	0.4105138
Five-hour ahead prediction ($p = 0.5$)				
I	0.2061686	0.5563662	0.2920678	0.3137478
II	0.2121247	0.3511250	0.4196604	0.4108466
III	0.2968155	0.2723984	0.2459617	0.2496851
IV	0.8465184	0.8381369	0.2263648	0.2067392
V	0.5262515	0.5552034	0.4143909	0.4171769
Ten-hour ahead prediction ($p = 0.5$)				
I	0.8685109	0.8011304	0.3142247	0.2812013
II	1.0237179	1.0336110	0.2768460	0.2840824
III	1.0044999	1.0019792	0.2015967	0.2013436
IV	0.6477370	0.6271771	0.2816179	0.4896072
V	0.9815581	0.9519172	0.4017764	0.4117593

Table XI: Accuracy of predicting the rate of incoming attacks against the victim network using the LRD-aware FARIMA and the LRD-less ARMA models and the measures specified in Section II. Basically, the smaller the value $\mathsf{PMAD}_{\mathrm{FARIMA}}$ ($\mathsf{PMAD}_{\mathrm{ARMA}}$), the more accurate the FARIMA (ARMA) prediction. $p = 0.5$ means that we start predicting in the midpoint of the length of each time period.

Second, Period III does not exhibit LRD. However, both the LRD-aware FARIMA and the LRD-less ARMA models can predict incoming attacks up to five hours ahead. Indeed, the prediction accuracy of FARIMA is slightly smaller than the accuracy of ARMA. This also confirms that if an attack process does not exhibit LRD, it is better not to use LRD-aware prediction models; if an attack process does exhibit LRD, it *can be* better to use LRD-aware prediction models.

Third, although Period IV exhibited LRD, even its one-hour ahead FARIMA prediction is not good enough, with over 50% error. While it is unclear what caused this fact, we note that the underestimation error for five hour ahead prediction is still reasonable for Period IV (22.6% for PMAD'_{FARIMA} and PMAD'_{ARMA}). This means that if one is willing to over-provisioning defense resource to some extent, then the prediction for Period IV is perhaps still useful.

Fourth, Period V resists both prediction models in terms of both overall accuracy and underestimation accuracy. The fundamental cause of this is unknown, and left for future studies. Nevertheless, we suspect that the Extreme Value Theorem could be exploited to solve this problem.

C. Exploring Cause of LRD

We also wanted to know whether or not the legitimate LRD phenomenon exhibited by the four attack super-processes against the victim network during Periods I, II, IV and V are caused by the superposition of underlying processes that exhibited the heavy-tail phenomenon. That is, we want to know how many attack processes — among the 166 attack processes corresponding to the victims during each of the four periods — exhibit the heavy-tail phenomenon. Using the methodology as in Section VI, we found that among the vector of $(166, 166, 166, 166)$ attack processes during Period I, II, IV and V, the vector of processes that exhibit the heavy-tail phenomenon is correspondingly $(101, 0, 24, 31)$. This means that Period I is the only period during which a majority of processes exhibited the heavy-tail phenomenon. A few or even none processes in the three other periods exhibited the heavy-tail phenomenon. This, again, suggests that the LRD phenomenon exhibited by the honeypot-captured data at a more macroscopic level probably does not share the cause that is believed for benign traffic [1].

D. Summary

By looking at the same attack data but from a more macroscopic perspective, we can draw similar findings as to the ones obtained when we look at the attack data from a more microscopic perspective. Specifically, among the five attack super-processes corresponding to the victim network during five periods, LRD was exhibited by four of them. We also found that LRD-aware models *can* — but not necessarily always — predict the incoming attacks more accurately than LRD-less models, and that LRD-less models are more appropriate for prediction in dealing with processes that do not exhibit LRD. Finally, we re-affirmed that LRD as exhibited by the honeypot-captured data is probably not caused by the superposition of attack processes that exhibit the heavy-tail phenomenon.

VIII. Limitations

This study has some limitations.

Data size

The dataset, albeit over $47 + 18 + 54 + 21 + 80 = 220$ days in total (five periods of time), only corresponds to 166 honeypot IP addresses. We wish to have access to significantly larger dataset of this kind. Nevertheless, it is notoriously difficult to get such data for various reasons. For example, it appears that even the PREDICT project (`https://www.predict.org/`) does not offer this kind of data. Still, this paper touches an important direction for cyber security research because of the potential reward in understanding the statistical properties of cyber attacks and in possibly predicting the incoming attacks with reasonable accuracy. The statistical analysis methodology used in the present paper would be useful to researchers who have access to larger datasets of this kind.

Low-interaction vs. high-interaction honeypot

The dataset is attack-agnostic in the sense that we know the ports/services the attackers attempt to attack, but to a large extent we do not know the specific attack details. This is mainly attributed to that the data was collected using low-interactive honeypots. Although this issue can be resolved with data that is collected using high-interactive honeypots, we are fully aware of the legitimate concerns about such studies from a legal perspective. Nevertheless, our methodology is equally applicable to the analysis of cyber attack data that is captured by high-interactive honeypots.

Honeypot vs. production network

The data was collected using honeypot rather than using production network. For real-life adoption of the prediction capability presented in the paper, attack traffic would be blended into the production traffic. Whether or not the blended traffic also exhibits the LRD phenomenon is an interesting future study topic. The main challenge here is to set up such a data collection instrument, which naturally incurs legitimate non-technical (e.g., legal or privacy) concerns. Indeed, we do not have access to such data. (It may not be a good idea to simply blend the honeypot traffic with benign production traffic because this might disrupt the attack process structure.)

IX. Conclusion and Future Work

We introduced the concept of stochastic cyber attack processes, which offers a new perspective for studying cyber attacks. We also proposed a statistical methodology for analyzing the processes. By applying the methodology to some honeypot-collected attack data, we found that majority of the attack processes exhibit the LRD (Long-Range Dependence) phenomenon. We demonstrated that by using LRD-aware models we can predict the attack rates five-hours ahead, especially when aggregating the victims into a network. This hints that attacks against enterprise-level networks are probably more predictable than attacks against individual computers. The power comes from “gray-box” (rather than “black-box”) prediction.

Future research

The present study introduces a range of interesting problems for future research. First, we need to further improve the prediction accuracy, even though our study showed that the FARIMA model can predict the attack process fairly well (especially when aggregating the victims into a network). For this purpose, we plan to use some advanced models with high volatilities.

Second, we hope to rigorously explain the fundamental cause of the LRD phenomenon as exhibited by the honeypot-captured attack data. As shown in the paper, the cause of LRD is mysterious, although we know that it is probably not caused by the superposition of processes that exhibit the heavy-tail phenomenon, and that it is probably not caused by the intense attacks launched by individual attackers.

Third, we observed that the attacks at the scale of individual IP addresses (as investigated in Sections IV-VI) and the attacks at the scale of victim networks (i.e., by aggregating the 166 victim IP addresses into a network as shown in Section VII) exhibit similar phenomena. This hints a sort of scale-invariance that, if turns out to hold, would have extremely important implications (for example) in achieving scalable analysis of cyber attacks. In order to resolve this issue, larger datasets are needed albeit extremely hard to obtain.

Acknowledgement This study was IRB-approved.

References

[1] S. Resnick, *Heavy-Tail Phenomena: Probabilistic and Statistical Modeling*. Springer, 2007.

[2] Z. Li, A. Goyal, Y. Chen, and V. Paxson, “Towards situational awareness of large-scale botnet probing events,” *Information Forensics and Security, IEEE Transactions on*, vol. 6, no. 1, pp. 175–188, march 2011.

[3] S. Almotairi, A. Clark, G. Mohay, and J. Zimmermann, “A technique for detecting new attacks in low-interaction honeypot traffic,” in *Proc. International Conference on Internet Monitoring and Protection*, 2009, pp. 7–13.

[4] S. I. Almotairi, A. J. Clark, G. M. Mohay, and J. Zimmermann, “Characterization of attackers’ activities in honeypot traffic using principal component analysis,” in *Proc. IFIP International Conference on Network and Parallel Computing*, 2008, pp. 147–154.

[5] A. Clark, M. Dacier, G. Mohay, F. Pouget, and J. Zimmermann, "Internet attack knowledge discovery via clusters and cliques of attack traces," *Journal of Information Assurance and Security*, vol. 1, no. 1, pp. 21–32, 2006.

[6] V. H. Pham, "Honeypot traces forensics by means of attack event identification," Ph.D. dissertation, Thesis, 09 2009.

[7] M. V. Mahoney and P. K. Chan, "Learning nonstationary models of normal network traffic for detecting novel attacks," in *Proc. 8th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'02)*, 2002, pp. 376–385.

[8] O. Thonnard and M. Dacier, "A framework for attack patterns' discovery in honeynet data," *Digital Investigation*, vol. 5, pp. S128–S139, 2008.

[9] F. Pouget and M. Dacier, "Honeypot-based forensics," in *AusCERT2004, AusCERT Asia Pacific Information technology Security Conference*, 05 2004.

[10] S. I. Almotairi, A. J. Clark, M. Dacier, C. Leita, G. M. Mohay, V. H. Pham, O. Thonnard, and J. Zimmermann, "Extracting inter-arrival time based behaviour from honeypot traffic using cliques," in *5th Australian Digital Forensics Conference*, 2007, pp. 79–87.

[11] G. Conti and K. Abdullah, "Passive visual fingerprinting of network attack tools," in *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, 2004, pp. 45–54.

[12] Y. Gao, Z. Li, and Y. Chen, "A dos resilient flow-level intrusion detection approach for high-speed networks," in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006, pp. 39–.

[13] M.-S. Kim, H.-J. Kang, S.-C. Hong, S.-H. Chung, and J. W. Hong, "A flow-based method for abnormal network traffic detection," in *NOMS (1)'04*, 2004, pp. 599–612.

[14] F. Dressler, W. Jaegers, and R. German, "Flow-based worm detection using correlated honeypot logs," *Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference*, pp. 1–6, 2007.

[15] T. Dubendorfer and B. Plattner, "Host behaviour based early detection of worm outbreaks in internet backbones," in *Proc. IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, 2005, pp. 166–171.

[16] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," 2008, vol. 36, pp. 1–24.

[17] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer, "Using machine learning techniques to identify botnet traffic," in *Proc. IEEE LCN Workshop on Network Security (WoNS'2006)*, 2006, pp. 967–974.

[18] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Trans. Netw.*, vol. 2, no. 1, pp. 1–15, 1994.

[19] W. E. Leland and D. V. Wilson, "High time-resolution measurement and analysis of lan traffic: Implications for lan interconnection," in *INFOCOM*, 1991, pp. 1360–1366.

[20] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level," *IEEE/ACM Trans. Netw.*, vol. 5, no. 1, pp. 71–86, 1997.

[21] G. Samorodnitsky, "Long range dependence," *Foundations and Trends in Stochastic Systems*, vol. 1, no. 3, pp. 163–257, 2006.

[22] J. Beran, *Statistics for Long-Memory Processes*. Chapman and Hall, 1994.

[23] W. Willinger, M. S. Taqqu, W. E. Leland, and V. Wilson, "Self-similarity in high-speed packet traffic: analysis and modeling of ethernet traffic measurements," *Statistical Sci.*, vol. 10, pp. 67–85, 1995.

[24] P. Abry and D. Veitch, "Wavelet analysis of long-range-dependent traffic," *IEEE Transactions on Information Theory*, vol. 44, no. 1, pp. 2–15, 1998.

[25] P. Embrechts, C. Kluppelberg, and T. Mikosch, *Modelling Extremal Events for Insurance and Finance*. Springer, Berlin, 1997.

[26] J. D. Cryer and K.-S. Chan, *Time Series Analysis With Applications in R*. New York: Springer, 2008.

[27] Z. Qu, "A test against spurious long memory," Boston University - Department of Economics, Boston University - Department of Economics - Working Papers Series WP2010-051, 2010.

[28] T. Mikosch and C. Starica, "Nonstationarities in financial time series, the long-range dependence, and the igarch effects," *The Review of Economics and Statistics*, vol. 86, no. 1, pp. 378–390, February 2004.

[29] X. Shao, "A simple test of changes in mean in the possible presence of long-range dependence," *Journal of Time Series Analysis*, vol. 32, no. 6, pp. 598–606, November 2011.

[30] $http : //dionaea.carnivore.it/$.

[31] $https : //alliance.mwcollect.org/$.

[32] $http : //amunhoney.sourceforge.net/$.

[33] P. Baecher, M. Koetter, M. Dornseif, and F. Freiling, "The nepenthes platform: An efficient approach to collect malware," in *In Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2006, pp. 165–184.

[34] S. Almotairi, A. Clark, G. Mohay, and J. Zimmermann, "Characterization of attackers' activities in honeypot traffic using principal component analysis," in *Proceedings of the 2008 IFIP International Conference on Network and Parallel Computing*, 2008, pp. 147–154.

[35] D. J. Daley and D. Vere-Jones, *An Introduction to the Theory of Point Processes, Volume 1 (2nd ed.)*. Springer, 2002.

[36] S. G.R. and W. J.A., *Empirical Processes with Applications to Statistics*. Springer, 1986.

[37] D. R.B. and S. M.A., *Tests Based on EDF Statistics*. Springer, 1986.

[38] M. Chandra, N. D. Singpurwalla, and M. A. Stephens,

"Kolmogorov statistics for tests of fit for the extreme value and weibull distributions," *J. Amer. Statist. Assoc.*, vol. 74, pp. 729–735, 1981.

[39] V. Choulakian and M. Stephens, "Goodness-of-fit tests for the generalized pareto distribution," *Technometrics*, vol. 43, pp. 478–484, 2001.

[40] Tsay, *Analysis of Financial Time Series*. Wiley, 2010.

[41] M. S. Taqqu, V. T. Teverovsky, and W. Willinger, "Estimators for long range dependence: An empirical study," *Fractals*, vol. 3, no. 4, pp. 785–798, 1995.

[42] W. Rea, M. Reale, and J. Brown, "Estimators for long range dependence: An empirical study," *arXiv: 0901.0762v1*, 2009.

APPENDIX

A. Methods for Estimating Hurst Parameter

We use seven major Hurst parameter estimation methods for evaluating LRD [22]. Note that one should not just report Hurst value based on one method, but should check several Hurst values based on different estimation methods [41], [42].

1) RS method. For a time series $\{X_t, t \geq 1\}$, with partial sum $Y_t = \sum_{i=1}^{t} X_i$, and sample variance $S_t^2 = \frac{1}{t}\sum_{i=1}^{t} X_i^2 - \left(\frac{1}{t}\right)^2 Y_t^2$, the R/S statistic is defined as

$$\frac{R}{S}(n) = \frac{1}{S_n}\left[\max_{0\leq t\leq n}\left(Y_t - \frac{t}{n}Y_n\right) - \min_{0\leq t\leq n}\left(Y_t - \frac{t}{n}Y_n\right)\right].$$

For LRD series, we have

$$\mathsf{E}\left[\frac{R}{S}(n)\right] \sim C_H n^H,$$

as $n \to \infty$, where C_H is a positive, finite constant independent of n.

2) AGV (Aggregated variance) method. Divide time series $\{X_t, t \geq 1\}$ into blocks of size m. The average within a block is

$$X^{(m)}(k) = \frac{1}{m}\sum_{i=(k-1)m+1}^{km} X_i, \quad k = 1, 2 \ldots.$$

Take the sample variance of $X^{(m)}(k)$, $k = 1, 2, \ldots$ within each block. The sample variance is an estimator of $\mathrm{Var}(X^{(m)})$. For LRD series, we have

$$\mathrm{Var}\left(X^{(m)}\right) \sim cm^{-\beta}, \quad m \to \infty,$$

where $\beta = 2H - 2$, and c is a finite positive constant independent of m.

3) Peng Method. The series is broken up into blocks of size m. Compute partial sums $Y(i)$, $i = 1, 2 \ldots, m$ within blocks. Fit a least-square line to the $Y(i)$'s and compute the sample variance of the residuals. This procedure is repeated for each of the blocks, and the resulting sample variances are averaged. It can be shown that the resulting number is proportional to m^{2H} for LRD series.

4) Per (Periodogram) Method. One first calculates

$$I(\lambda) = \frac{1}{2\pi N}\left|\sum_{j=1}^{N} X_j e^{ij\lambda}\right|,$$

where λ is the frequency, N is the number of terms in the series, and X_j is the data. A series with LRD should have a periodogram, which is proportional to λ^{1-2H} for $\lambda \approx 0$. Therefore, a regression of the logarithm of the periodogram on the logarithm of the frequency gives coefficient $1 - 2H$.

5) Box (Boxed Periodogram) method. The boxed periodogram method was developed specifically to deal with the problem of having most of the points, which are used to estimate H, on the right-hand side of the graph.

6) Wave (Wavelet) method. Wavelets can be thought of as akin to Fourier series but using waveforms other than sine waves. The estimator used here fits a straight line to a frequency spectrum derived using wavelets [24].

B. Heavy-tail Distributions

A random variable X is said to belong to the Maximum Domain of Attraction (MDA) of the extreme value distribution H_ξ if there exists constants $c_n \in \mathbb{R}_+$, $d_n \in \mathbb{R}$ such that its distribution function F that satisfies

$$\lim_{n\to\infty} nF(c_n x + d_n) = H_\xi(x).$$

It is known that for $\xi = 1/\alpha > 0$, $F \in \mathrm{MDA}(H_\xi)$ is equivalent to the distribution function satisfies

$$F(x) = 1 - x^{-\alpha}L(x),$$

where $L(x)$ is a slow varying function L (cf. [25]). In statistics, X is said to follow a heavy-tailed distribution if $F \in \mathrm{MDA}(H_\xi)$. Note that [1] if X has a heavy-tailed distribution, then its tail

$$\frac{\bar{F}(cy)}{\bar{F}(c)} \approx y^{-\alpha}, y > 0, c \to \infty,$$

where $\bar{F}(c) = 1 - F(c)$. There are many methods for estimating the parameter α [25], [1]. In the following, we briefly review the widely-used method, called Point Over Threshold (POT).

Let $X_1, \ldots, X_n$ be independent and identically distributed random variables from $F \in \mathrm{MDA}(H_\xi)$, then we may choose a high threshold u such that

$$\lim_{u\to x_F}\sup_{0<x<x_F-u}|\bar{F}_u(x) - \bar{G}_{\xi,\beta(\mu)}(x)| = 0,$$

where x_F is the right end poind point of X, and

$$F_u(x) = P(X - u \leq x | X > u), \quad x \geq 0,$$

and $\bar{G}_{\xi,\beta(\mu)} = 1 - G_{\xi,\beta(\mu)}$ is the survival function of generalized Pareto distribution (GPD)

$$\bar{G}_{\xi,\beta(\mu)}(x) = \begin{cases} \left(1 + \xi\frac{x}{\beta}\right)^{-1/\xi}, & \xi \neq 0, \\ \exp\{-x/\beta\}, & \xi = 0. \end{cases},$$

where $x \in \mathbb{R}^+$ if $\xi \in \mathbb{R}^+$, and $x \in [0, -\beta/\xi]$ if $\xi \in \mathbb{R}^-$. ξ and β are called shape and scale parameters. The POT method states that if $X_1, \ldots, X_n$ are heavy-tailed data, then $[X_i - u | X_i > u]$ follows a generalized Pareto distribution.

C. Goodness-of-fit Test Statistics

In statistics, the goodness-of-fit of a distribution describes how well a distribution fits a set of observations. We use three commonly used goodness-of-fit test statistics: Kolmogorov-Smirnov (KS), Anderson-Darling (AD) and Cramér-von Mises (CM).

Let $X_1, \ldots, X_n$ be independent and identical random variables with distribution F. The empirical distribution F_n is defined as

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{I}(X_i \le x),$$

where $\mathrm{I}(X_i \le x)$ is the indicator function:

$$\mathrm{I}(X_i \le x) = \begin{cases} 1, & X_i \le x, \\ 0, & o/w. \end{cases}$$

The KS test statistic is defined as

$$\mathrm{KS} = \sqrt{n} \sup_x |F_n(x) - F(x)|,$$

and the CM test statistic is defined as

$$\mathrm{CM} = n \int (F_n(x) - F(x))^2 dF(x).$$

The AD test statistic is defined as

$$\mathrm{AD} = n \int (F_n(x) - F(x))^2 w(x) dF(x),$$

where $w(x) = [F(x)(1 - F(x))]^{-1}$. Note that AD test is a variant of CM test by giving more weights to the observations in the tail of the distribution, which is useful in detecting outliers. Based on samples, the CM and AD test statistics can be computed as follows.

- Estimate the unknown parameter(s) for distribution F, and make the transform $u_{(i)} = F(x_{(i)})$, for $i = 1, \ldots, n$, where $x_1 \le x_{(2)} \le \ldots \le x_{(n)}$.
- Calculate CM and AD as follows [36], [37]:

$$\mathrm{CM} = \sum_{i=1}^{n} \left[u_{(i)} - \frac{2i-1}{2n} \right]^2 + \frac{1}{12n},$$

and

$$\mathrm{AD} = -n - \frac{1}{n} \sum_{i=1}^{n} (2i-1) \left[\log(u_{(i)}) - \log(1 - u_{(n+1-i)}) \right].$$

Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures

Erhan J. Kartaltepe[1], Jose Andre Morales[1], Shouhuai Xu[2], Ravi Sandhu[1]

[1] Institute for Cyber Security, University of Texas at San Antonio
{erhan.kartaltepe,jose.morales,ravi.sandhu}@utsa.edu
[2] Department of Computer Science, University of Texas at San Antonio
shxu@cs.utsa.edu

Abstract. Botnets have become a major threat in cyberspace. In order to effectively combat botnets, we need to understand a botnet's Command-and-Control (C&C), which is challenging because C&C strategies and methods evolve rapidly. Very recently, botmasters have begun to exploit social network websites (e.g., `Twitter.com`) as their C&C infrastructures, which turns out to be quite stealthy because it is hard to distinguish the C&C activities from the normal social networking traffic. In this paper, we study the problem of using social networks as botnet C&C infrastructures. Treating as a starting point the current generation of social network-based botnet C&C, we envision the evolution of such C&C methods and explore social networks-based countermeasures.

Keywords: Botnet, command-and-control, social networks, security.

1 Introduction

The critical difference between botnets and other malware is that botmasters use a Command-and-Control (C&C) to coordinate large numbers of individual bots (i.e., compromised computers) to launch potentially much more damaging attacks. Botmasters also evolve their C&C strategies and methods very rapidly to evade defenders' countermeasures. Therefore, from a defender's perspective, it is always important to understand the trends and practices of botnet C&C [8, 15, 19, 23, 29, 11, 26]. Previous studies have mainly focused on two approaches: *host-centric* [31, 36] and *network-centric* [16, 18, 17, 3, 7, 8, 14, 23, 20, 6]. The host-centric approach aims to detect suspicious host activities, such as the use of incoming network data as system call arguments. The network-centric approach attempts to detect suspicious network activities by (for example) identifying network traffic patterns. The fact that a social network-based botnet C&C on `Twitter.com` was detected by "digging around" [25] suggests that we need to pursue more detection approaches.

Our contributions. Through an *application-centric* approach, we study the problem of botnets that use social network websites as their C&C infrastructures. First, we characterize the current-generation of social network-based botnet C&C, describing their strengths and weaknesses. Our characterization, while inspired by [25], is broader and deeper. Second, we envision how current social network-based botnet C&C might evolve in the near future, which capitalize on their strengths while diminishing their weaknesses. Third, we explore countermeasures for dealing with both current and future generations of social network-based botnet C&C. Since social network providers as well as client machines are victims of a social network-based botnet C&C, both server-side and client-side countermeasures are demonstrated and tested for both effectiveness and performance. Fourth, we discuss the limitations of the application-centric approach demonstrated in this paper, which suggests the need to integrate it with the aforementioned host-centric and network-centric methods because the three approaches are complementary to each other.

Paper outline. Section 2 discusses related prior work. Section 3 presents a characterization of the current generation of social network-based botnet C&C. Section 4 envisions the next-generation of social network-based botnet C&C. Sections 5 and 6 investigate server-end and client-end solutions to detecting social network-based botnet C&C, respectively. Section 7 discusses how to integrate them and how they can benefit from host-centric and network-centric approaches. Section 8 describes future work and concludes the paper.

2 Related Work

Network-centric approach. This approach aims to detect botnets by correlating the network traffic of a computer population, including destination IP addresses, server names, packet content, event sequences, crowd responses, protocol graphs and spatial-temporal relationships [16, 18, 17, 3, 7, 8, 14, 23, 20, 6]. This approach is especially useful when only network traffic data is available.

Host-centric approach. This approach aims to differentiate malicious from benign processes running on host computers by observing that bot processes often use data received from the network as parameters in system calls [31]. A detection technique based on a high rate of failed connection attempts to remote hosts was recently presented in [36], which does not necessarily apply to the type of botnets we consider in the present paper because the connections are to popular social networking sites and are generally successful. This approach often looks deeply into the software stack [36].

Application-centric approach. This approach looks into the application-specific interactions. Previously, the focus has been put on IRC-based botnets (see, e.g., [14]). Recently, the possibility of exploiting emails as a botnet C&C was investigated [30], and the feasibility of detecting such botnets through their resulting spam traffic was presented in [35, 34, 37, 22]. In this paper we consider a specific class of applications, namely web-based social networking. The concept of social network-based botnet C&Cs can be dated back to 2007 [21, 12, 28, 13], but such botnets became a reality only very recently [2, 25]. In particular, [25] served as the starting point of the present study. It should be noted that the focus of the present paper is only remotely related to the abuse of social networks for other purposes [1].

3 Characterizing Current Social Network-Based Botnet

How Does Current Social Network-Based Botnet C&C Work? Jose Narario first reported the actual use of social network as a botnet C&C [25], although the concept had been proposed as early as 2007 [21, 28]. We call such a bot Naz, after its discoverer. At a high-level, Naz's botnet used accounts with the name upd4t3 owned by the botmaster on social network sites Twitter.com and Jaiku.com. These bots received Base64-encoded commands from these accounts to download malicious payloads onto the victimized bot computers. Since then, other C&Cs were discovered with variations of the botnet's scheme, such as the Twitter.com account Botn3tControl, which was shutdown days later.

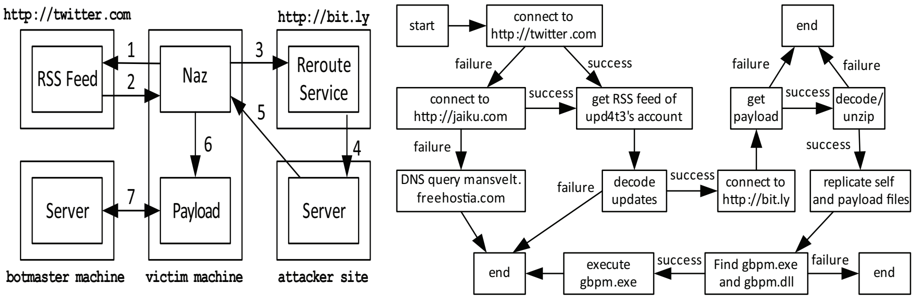


Fig. 1. Naz's C&C attack flow (left) and control flow (right)

To understand the behavior of Naz's botnet, we conducted two experiments. The first confirmed and extended Naz's C&C flow reported in [25]. Specifically, we obtained and ran a Naz sample on a machine by replacing its references to Twitter.com with a server under our control. This was necessary because the Twitter.com account was shutdown by Twitter.com's administrators shortly after its detection. We used our own Base64-encoded messages with a bit.ly URL we set up that redirected to the payload stored on our server. The payload was a Base64-encoded, compressed archive containing two files: gbpm.exe and gbpm.dll. These files were the originally identified payload package of Naz. In our analysis, we reconstructed the original C&C flows described below and depicted in the left-hand side of Figure 1.

1. The bot makes a HTTP GET request to upd4t3's RSS (Really Simple Syndication) feed at Twitter.com.
2. Twitter.com returns the RSS feed, containing Base64-encoded text in the description nodes.
3. The bot decodes the Base64-encoded text, revealing one or more bit.ly URLs, and makes a HTTP GET request to each. The bit.ly website provides short aliases for long URLs.
4. Each bit.ly URL redirects to a malicious zip file hosted on an independent attack server.

5. The bot downloads the malicious zip file as a payload.
6. The bot decodes and unzips the payload, replicates itself and the payload's uncompressed files, and then executes the payload's contents.
7. The payload attempts to gather user information from the victim computer and send it to a server selected by the botmaster.

To have a deeper understanding of the internal control flow of Naz, we conducted further black-box testing using data provided by Network Monitor [9] and CWSandbox [10], from which we draw the control flow details on the right-hand side of Figure 1. We observed that the bot made a copy of itself and the two files mentioned above in a temporary directory, and that when executing `gbpm.exe`, the bot dynamically injected code into `gbpm.exe`'s process. Moreover, we observed that Naz handled unexpected inputs as follows:

1. When we provided a URL to a bogus RSS feed, the bot failed to connect and instead attempted to access an RSS feed from a `Jaiku.com` account. This account name was hardwired in the bot program and had been deactivated by `Jaiku.com`'s administrator. This second connection failure led the bot to issue a DNS query on the domain name `mansvelt.freehostia.com`, after which the bot stopped producing network traffic. The site `mansvelt.freehostia.com` is currently unregistered and has no IP address.
2. When we placed plaintext sentences and URLs in our in-house RSS feed, the bot read the RSS feeds but did not show evidence of decoding and using the text in connection attempts.
3. When we modified the payload file to Base64-encoded only, compressed only, and replaced it with an executable file, the bot did not attempt to unzip or execute the file's contents. When we renamed the two payload files `gbpm.exe` and `gbpm.dll`, the bot did not attempt to execute the renamed `gbpm.exe` file, implying that the bot was program name sensitive.

Finally, it is interesting to note that a dynamic analysis of `gbpm.dll` revealed that the payload attempted to connect to a bank in Brazil. Moreover, both the analysis in [25] and our independent experiments demonstrate that Naz's botnet C&C serves primarily as a Trojan downloader [32].

Strengths of Naz's Botnet C&C. Naz's botnet C&C has the following advantages when compared with other botnet C&C infrastructures and methods:

1. Abusing trusted popular websites as a C&C server. Social networks and Web 2.0 sites such as `Twitter.com`, `FaceBook.com`, `LinkedIn.com`, and `YouTube.com` are not only legitimate, with verifiable SSL or EV-SSL certificates, but also heavily used by millions of users. Due to this heavy usage, light occasional traffic to one or more accounts is unlikely to be noticed compared to a user's actual traffic pattern. This avoids any unnecessary and sometimes suspicious DNS queries (e.g., for non-popular DNS names).
2. Exploiting popular port(s) for C&C communication. Port 80 is the de facto standard for web traffic, and most network traffic will flow through it. This helps bots blend in with benign traffic.

3. Abusing application features for automatic C&C. The botmaster uses application features, such as RSS feeds, to automatically update bots. Moreover, the commands are so light-weight that they cannot be easily discerned from normal social network traffic.

The above discussion demonstrates that botmasters have begun to exploit "*hiding in plain sight*" to conduct stealthy botnet C&C. By piggybacking on the reputation and legitimacy of social network websites, botnet C&C activities may remain hidden, while defeating the "many eyes" defense [30].

Weaknesses of Naz's Botnet C&C. We need to understand the weaknesses of current generation of social network-based botnet C&Cs because these weaknesses will likely be absent in future generations. This is not meant to help the attackers, rather it is meant to help the defenders look ahead. Our examination shows that Naz's botnet C&C has weaknesses, which are omitted due to space limitation.

4 Envisioning Future Social Network-Based Botnet

In order to defeat future social network-based botnets, we must think ahead of the attackers. For this purpose, we can show how the aforementioned weaknesses of the current generation of social networks-based botnet C&Cs can be avoided. Due to space limitation, details are omitted.

5 Server-side Countermeasures

5.1 The Detection Mechanism

A key observation behind our detection mechanism is that, regardless of the channel, provider, or account, social network messages are in text. Thus, if botmasters want to use social networks for their C&C, they would encode their commands textually. Moreover, just like legitimate messages may include web links, so might C&C messages (e.g., links for downloading payload). These observations inspired us to distinguish between encoded and plain texts and to follow unencoded links to their destination. Our detection mechanism can be adopted by the webserver as shown in Figure 2, and the resulting system would operate as follows (with steps 2 and 3 relevant to our countermeasure):

1. Alice logs into her social network and updates her status display using a content form.
2. The social network's content updater sends the text content to our server-end system.
3. The detection mechanism, which will be implemented as a classifier in our prototype system, determines if the text is suspicious and returns a result.
4. The content form updates the database with the message and whether it was marked suspicious.

5. Bob check's Alice content display, either through a feed like RSS or by visiting the site.
6. The content display requests the content from the database.
7. The database returns non-suspicious content; if a threshold level of suspicious messages (determined by policy) has been reached, the database returns a "suspicious account" message.
8. The content display shows those retrieved messages to the user, or a "suspicious account" message if the suspicious message threshold has been reached.

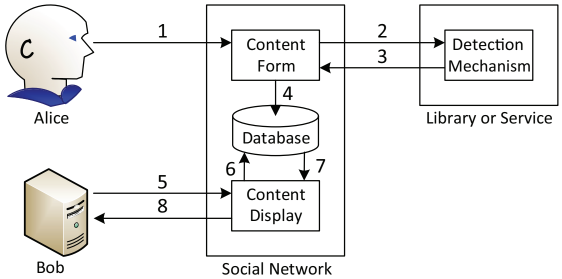


Fig. 2. Example scenario using our server-side detection mechanism

The server-side detection mechanism has the following advantages. First, it is *account agnostic* because it looks for text attributes that are shared with encoded text rather than individual behavioral patterns. Second, it is *language agnostic* because it looks at text for attributes that are shared with encoded text rather than individual words. As a result, the detection mechanism is effective for any language using Roman characters (English, Spanish, French, etc.). Third, it is *easy to deploy* because it can take advantage of light-weight machine learning algorithms and thus make decisions in real-time. Moreover, the code is easy to deploy as a library or software-as-a-service. Fourth, it can *follow unencoded links* to determine if the destination is a trusted source, say by using SSL authentication as a trust infrastructure. In the next two subsections we analyze the effectiveness and performance of our approach.

5.2 Prototype Implementation and its Effectiveness and Limitations

Prototype implementation. To demonstrate the effectiveness of our system, we instantiated the detection mechanism as `Weka`'s [33] J48 decision tree algorithm (because it is quick and readily usable, but other tailored algorithms can be used instead in a plug-and-play fashion) to classify input messages so as to distinguish between Base64- or Hexadecimal-encoded text and natural language text. For links in the clear, by following links to their destination, we can mark the content as "suspicious" if it is an atypical file (e.g., an executable, library, encoded, or compressed file, or a combination of these). To build a pool of "non-suspicious" text, we screenscraped 200 `Twitter.com` accounts to build a list of 4000 messages. Our pool of bot commands were 400 short random commands of fifteen to thirty characters that were then encrypted using RC4 stream cipher and then encoded, giving a 10:1 set of normal to suspicious text. We then split the messages into a training set with 70% of both types and a test set with the remaining 30%. Recognizing that altering the natural Base64 or Hexadecimal alphabet with alternate characters such as spaces or punctuation could be used to obfuscate the text, we also ran our classifier against such alternate encoding schemes.

Effectiveness. For standard Base64 and Hexadecimal encoding schemes, our classifier was able to quickly distinguish between our "normal" and "suspicious" text samples in an account-agnostic way, no false positives and no false negatives, for both Base64 and Hexadecimal encoding (see Table 1). Moreover, our classifier maintained this accuracy even when the commands were obfuscated with other words—the distinctiveness of the encoded commands was readily apparent. The results were so perfect because the attributes we used—number of spaces in the text, longest word, and shortest word—cleanly divided the "normal" and "suspicious" text. To produce non-standard Base64 and Hexadecimal encoding schemes, we randomly swapped ten of the standard alphabet with alternate ones from a pool of space and punctuation characters. Our profiler was able to distinguish between them in an account-agnostic way, with a false positive rate of 0.0% and false negative rate of 1.25% for Alternate Base64 encodings, and a false positive rate of 3.25% and false negative rate of 12.5% for Alternate Hexadecimal encodings (see Table 1).

	Base64		**Hexadecimal**		**Alt. Base64**		**Alt. Hexadecimal**	
	Actual Positive	Actual Negative	Actual Positive	Actual Negative	Actual Positive	Actual Negative	Actual Positive	Actual Negative
Tested Positive	100%	0%	100%	0%	100%	1.25%	96.75%	12.5%
Tested Negative	0%	100%	0%	100%	0%	98.75%	3.25%	87.5%

Table 1. Results with respect to various Base64 and Hexadecimal encodings

The classifier's accuracy dropped significantly when the commands were obfuscated with other words, especially with Hexadecimal encoding schemes and with spaces as alternate characters. We note that with such an encoding mechanism, *a priori* knowledge of words or characters to excise from the message would be necessary to extract the non-command content from the meaningful botnet commands. This form of steganography is essentially indistinguishable from typical steganography, where a botmaster would hide the bot commands in such a way as to not attract attention to themselves, i.e., using natural language words as code for commands or URLs.

Limitations. Hiding commands in a social network-based C&C using steganography makes it difficult for programs or even humans to identify the presence of a command within a message. Since social network messages left by users are unstructured content, a crafty adversary can hide a bot command within a message in such a way that a human reading the message could not identify the message as a command. Combined with encryption, reverse analysis—even with a captured bot—may not yield the interpreted commands. Thus, running a steganographic reversal algorithm on a C&C message would not return data that was a clear bot command. [30]. However, our server-side solution coupled with a client-side counterpart that detects when a process is acting on input from a social network-based C&C would provide a complete solution to this emerging threat. In Section 7 we will discuss how these limitations may be overcome in a bigger solution framework.

5.3 Performance

Evaluation methodology and environment setup. In order to demonstrate the efficiency of our server-side detection mechanism, we measured the performance of our prototype. We implemented it into CompactSocial, a microblogging service that emulates the constraints of `Twitter.com`. CompactSocial provides a simple interface to both update a status message and view any account's messages using a web browser. Moreover, an auto-updated RSS feed contains the text of the last ten account updates. CompactSocial was written in Java 6, update 11 and ran as a web application deployed to Apache Tomcat 6.0.20. When used as a library, our server-end solution was deployed as a `.jar` file; when deployed as a service, the classifier ran as a stand-alone web application deployed to Apache Tomcat 6.0.20. The classifier and CompactSocial used shared cryptographic keys for authentication. For processing incoming and outgoing messages, Javas crypto library was used to compute any hash value or HMAC it needed, in both cases using the SHA1 algorithm.

The system environment is depicted in Figure 3. Both servers reside within a university campus network, and the CompactSocial clients are both within and outside the campus network. The CompactSocial and text-classifier servers are called `mercury` and `apollo`, respectively. There are two CompactSocial client machines: `minerva` acted as an external computer within the LAN with authorized access to `mercury` through a simple CompactSocial client, and `mars` was an adversary client machine within the campus network, employing Naz+ to read updates made by `minerva`. A fifth machine, `venus` mimicked the `minerva`'s functionality and tested the performance of the classifier on a non-dedicated internet

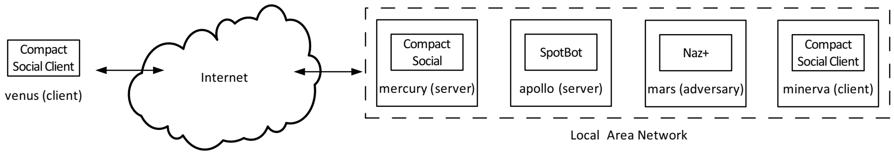


Fig. 3. Integrating the server-side solution into real life systems

connection. The three servers, hermes, jupiter, and euclid recognized each other by sharing some pair-wise keys. Table 2 reviews the concrete configurations of the machines and networks.

Machine	**Internet Connection**	**Relevant Software**
mercury	Gigabit LAN	Apache Tomcat 6.0.20, Sun JVM, CompactSocial
apollo	Gigabit LAN	Apache Tomcat 6.0.20, Sun JVM, classifier service
minerva	Gigabit LAN	Firefox 3.5, CompactSocial client
mars	Gigabit LAN	.NET 3.5 Framework, Naz+
venus	100 Megabit Cable	Firefox 3.5, CompactSocial client

Table 2. System settings (all machines use Intel Core 2 Duo, 2.93 GHz processor)

A CompactSocial client was developed in JavaScript to simulate a user updating their status in the CompactSocial web application. The CompactSocial client was developed as an addon and installed into Mozilla Firefox. Additionally, Naz+ was developed as a Windows service and written in C#, targeting the Microsoft .NET Framework, version 3.5. Naz+ periodically checked the RSS feed for CompactSocial test account, parsed the XML for the description node which contained the bot command as encrypted text (using a shared key with `mars` who updated the status message), and executed the command.

Performance benchmarking. To examine the delay incurred by the classifier when utilized by CompactSocial, time was marked before and after each transaction over 30000 requests at varying rates. We repeated this test ten times and took the average over the runs. Figure 4 shows the results over varying requests per second when the classifier was used as a library and a service.

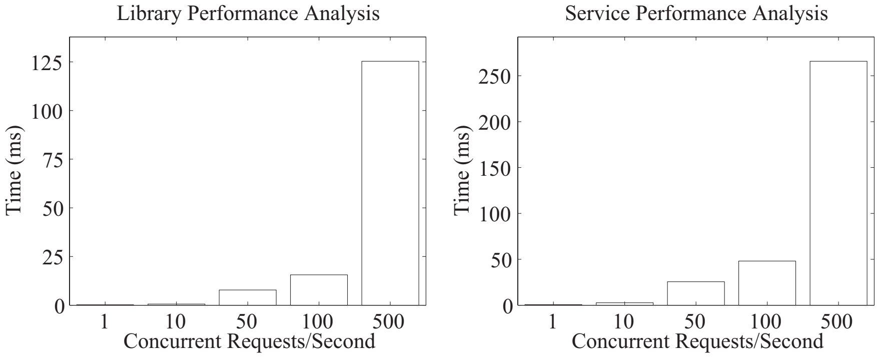


Fig. 4. Classifier library (left) and service (right) performance analysis

When used as a library, our classifier performed roughly twice as fast than its service counterpart, since no network traffic was required. At even 500 requests per second, the classifier handled all requests without incident. In practice, a large-scale enterprise would use our server-side classifier as a service, whether as an in-house server to make requests or a pay-as-you-go service to a third party. In our preliminary testing, the classifier service handles 500 requests per second on a non-dedicated machine with cycles to spare. In the advent that a larger throughput was necessary, a load balancer can reduce the requests for a particular machine to a manageable level.

Meeting the needs of real-life systems. For a social network provider like Twitter.com, that has fifteen million users, and is increasing at roughly one million users a month, we wondered how our classifier would stack up. A large percentage (85.3%) of Twitter.com accounts post less than once a day, whether due to lost interest or not having much to post. We classify these accounts as "passive users". The vast majority (99.5%) of accounts post less than sixteen messages daily, which would be a class of "active" users, who post regularly about events. The remaining 0.5% post more than sixteen updates, although Twitter.com restricts accounts to a 1000 updates per day limit. These are particularly engaged users or are shared accounts by multiple people in an organization posting under a common account. We classify these users as "explosive" users [5].

If all accounts hit their ceilings on Twitter.com, we'd have the posting rates in Table 3. With fifteen million users, Twitter.com would handle a ceiling of 1410.7 messages per second (in actuality, most users do not hit these ceilings, so the actual threshold is far less). In this worst case, assuming the one million account per month growth rate and the same distribution of account usage, Twitter.com will accommodate an uptick per month of nearly 100 messages per second.

15000000 users	Percentage	Update Rate	Messages Per Day	Messages Per Second
Passive users	85.3%	1/day	12795000	148.1
Active users	14.2%	16/day	34080000	394.5
Explosive users	0.5%	1000/day	75000000	868.1
Total users	100%	—	121875000	1410.7

Table 3. Twitter.com usage analysis

Because our server-side approach is account agnostic, it does not need to build an account history for each user and as a result, would only need to check a few messages to determine if the account is being used in a suspicious way. Given the above scenario and a policy that checks periodically verifies one message daily and the first three messages for a new user's account, then our classifier would only need to check fifteen million messages per day, or 173.7 messages per second, with a monthly increase of 12.1 messages per second. If only active and explosive accounts were targeted (which would be more likely behavior for a botnet C&C), this would decrease to 25.6 messages per second with an increase of 2.1 messages per second. Thus, even with these simple non-discriminating policies and worst-case Twitter.com usage scenario, our classifier as a service can handle an enterprise-level throughput of requests, and different policy strategies may be employed to throttle down the throughput further.

6 Client-side Countermeasures

6.1 The Detection Mechanism

Detection attributes. We propose detecting social network-based botnet C&Cs using three attributes: self-concealing, dubious network traffic, and unreliable provenance.

- Self-Concealing: A self-concealing process is one which attempts to avoid detection with the use of stealth mechanisms. We consider two specific instances of this type:

  Graphical User Interface Many applications that read RSS feeds interact with a user via a graphical user interface. Bots and other malware will attempt to avoid detection, and as a result may run in the background as a service or hidden process without an explicit interface. A process without a graphical user interface can be identified as possibly self-concealing.

  Human Computer Interaction Most benign software works by reacting to user input via a keyboard or mouse. Malware processes tend to run hidden and independent of user input and don't require explicit keyboard or mouse events provided by a user to perform a nefarious act. A process without human/computer interaction can be identified as possibly self-concealing.
- Dubious Network Traffic: A process with dubious network traffic is one which engages in network communication with another machine in a covert or devious way. We consider three specific instances of this type:

  Social Network Request Exclusively visiting social networking sites is not suspicious; however, social network-based botnet C&C craftily abuse the popularity and good name of social networking sites; thus, exclusive requests to social networking or web 2.0 sites is considered a possible trigger event for dubious network traffic.

  Encoded Text Processing Since social network-based bots read commands as encoded text, processes making connections to sources that provide encoded or encrypted text is anomalous. Accepting connections with encoded text and processing it by decoding or decrypting it can be considered as possibly dubious network traffic.

  Suspicious File Downloading In general, applications do not download suspicious files such as executable, library, compressed, or encoded/encrypted files without permission (though they may download image or text files). Social network-based C&C bot processes, on the other hand, act as Trojan downloaders and almost exclusively save executables or DLLs to the filesystem as malicious payload. Downloading such suspicious files can be considered as dubious network traffic.

- Unreliable Provenance: A process with unreliable provenance is one which lacks a reliable origin. We consider three specific instances:
  - Self-Reference Replication This is a feature malware uses to survive disinfection on a host machine, occurring when a process copies itself into a newly created file or an existent file (by modifying it) on the file system [24]. An installed file with its installer not having a verified signature can be identified as possibly having an unreliable provenance.
  - Dynamic Code Injection This is used by malware to insert malicious code into the memory space of an active process. Its end goal is to modify the process to perform nefarious deeds, possibly by piggybacking on that application's authorization settings. A process whose injector lacks a digital signature can be identified as possibly having an unreliable provenance since the injector's origin cannot be established.
  - Verifiable Digital Signature Digital signatures may be considered a hallmark of trust between users and well established software. Most organizations that publish software provide a signature for their program and related files. Malware authors typically do not employ digital signatures; as a consequence, a process running without a verifiable digital signature can be identified as possibly having an unreliable provenance.

Detection model. We say a process P has the self-concealing attribute (P_{sc}) if it lacks a graphical user interface ($P_{gui} = \mathsf{false}$) and does not accept human computer interaction ($P_{hci} = \mathsf{false}$). More formally,

$$(\neg P_{gui}) \wedge (\neg P_{hci}) \rightarrow P_{sc}.$$

We say a process P has the dubious network traffic attribute (P_{dnt}) if it performs social network requests ($P_{snr} = \mathsf{true}$) and encoded text processing ($P_{etp} = \mathsf{true}$) or does suspicious file downloading ($P_{sfd} = \mathsf{true}$) (or both). More formally,

$$P_{snr} \wedge (P_{etp} \vee P_{sfd}) \rightarrow P_{dnt}.$$

We say a process P has the unreliable provenance attribute (P_{up}) if it performs self-reference replication ($P_{srr} = \mathsf{true}$) or does dynamic code injection ($P_{dci} = \mathsf{true}$), and also lacks a verified digital signature ($P_{vds} = \mathsf{false}$). More formally,

$$(P_{srr} \vee P_{dci}) \wedge (\neg P_{vds}) \rightarrow P_{up}.$$

Putting the above altogether, we classify a process P as being suspicious of being a social network-based bot C&C process (P_{snbb}) if it is either self-concealing ($P_{sc} = \mathsf{true}$) or has an unreliable provenance ($P_{up} = \mathsf{true}$) (or both), and engages in dubious network traffic ($P_{dnt} = \mathsf{true}$). More formally,

$$(P_{sc} \vee P_{up}) \wedge P_{dnt} \rightarrow P_{snbb}.$$

6.2 Effectiveness and Limitations

Evaluation methodology and environment setup. To examine the effectiveness of the detection model described above, we collected data with respect to our detection attributes for both benign and malicious processes in order to distinguish them. For this purpose, we considered eighteen benign applications, four traditional bots, and the malicious Naz and prototype Naz+ bots (listed in Table 4). To provide a wide breadth, the benign applications are a broad selection of the most popular web browsers, RSS aggregators, Twitter clients, and RSS aggregators which read subscription feeds. Testing was performed using VMWare Workstation running Microsoft Windows SP3 using NAT for Internet access. Each application was executed separately for a period of four hours, followed by post-analysis. During testing of the eighteen benign applications, we interacted with each application by subscribing to and viewing different RSS feeds; attempting to subscribe to bogus RSS feeds, updating all RSS feeds every hour, reading individual feed articles. These tests were done to provide a wide range of expected and unexpected scenarios for each application to deal with while recording their behavior. In addition, we used a number of sensors to gather information about each process. Tracing of the three detection attributes described in Section 6.1 occurred from the moment a process starts executing.

Application	**Type**	**Application**	**Type**	**Application**	**Type**
AOL Explorer	Web Browser	Internet Explorer	Web Browser	RSS Bandit	RSS Aggregator
Avant	Web Browser	K-Meleon	Web Browser	RSS Owl	RSS Aggregator
Bobax	Traditional Bot	Maxthon	Web Browser	SeaMonkey	Web Browser
BlogBridge	RSS Aggregator	Mercury	RSS Aggregator	Snarfer	RSS Aggregator
FeedDemon	RSS Aggregator	Naz	SN-Based Bot	Tweetdeck	Twitter Client
FireFox	Web Browser	Naz+	SN-Based Bot	Twhirl	Twitter Client
Flock	Web Browser	Opera	Web Browser	Virut	Traditional Bot
Google Chrome	Web Browser	Ozdok	Traditional Bot	Waledac	Traditional Bot

Table 4. Client-side test set ("SN-Based Bot" stands for "Social Network-Based Bot")

Network traffic was collected using Windows Network Monitor [9]. Keyboard and mouse input was collected with a modified version of GlobalHook. Digital signatures were verified using SigCheck. Self-reference replication and dynamic code injection were accomplished with kernel hooks implementing known techniques [24]. The presence of a user interface was recorded by observing the creation of any window upon executing each application using EasyHook. User input, network traffic, graphical user interface interaction, self-reference replication and dynamic code injection were all recorded in real-time. For Virut and Waledac, static analysis and previous executions of these bots yielded their dubious network traffic results. Digital signatures were verified after the four hour testing of each process.

Effectiveness. The results are summarized in Table 5 and highlight some observations below. First, we observe that all benign applications lacked the self-concealing attribute as they all utilized a graphical user interface and accepted inputs from the user, such as reading an article, following a link, or updating an RSS feed. All bots but Virut demonstrated the self-concealing attribute since they did not have a graphical user interface or accept user input, although it appears that the command prompt window Virut displays may be accidental.

Application	Self-Concealing		Unreliable Provenance			Dubious Network Traffic			Result
	Graphical User Interface	Human Computer Interaction	Self-Reference Replication	Dynamic Code Injection	Verifiable Digital Signature	Social Network Request	Encoded Text Processing	Suspicious File Download	Social Network-Based Bot?
AOL Explorer	Y	Y	N	N	Y	N	N	N	N
Avant	Y	Y	N	N	Y	N	N	N	N
BlogBridge	Y	Y	N	N	Y	N	N	N	N
FeedReader	Y	Y	N	N	Y	N	N	N	N
Firefox	Y	Y	N	N	Y	N	N	N	N
Flock	Y	Y	N	N	Y	N	N	N	N
IE	Y	Y	N	N	Y	N	N	N	N
Chrome	Y	Y	N	N	Y	N	N	N	N
K-Meleon	Y	Y	N	N	Y	N	N	N	N
Maxthon	Y	Y	N	N	Y	N	N	N	N
Mercury	Y	Y	N	N	Y	N	N	N	N
Opera	Y	Y	N	N	Y	N	N	N	N
RSS Bandit	Y	Y	N	N	Y	N	N	N	N
RSS Owl	Y	Y	N	N	Y	N	N	N	N
SeaMonkey	Y	Y	N	N	Y	N	N	N	N
Snarfer	Y	Y	N	N	Y	N	N	N	N
Tweetdeck	Y	Y	N	N	N	Y	N	N	N
Twhirl	Y	Y	N	N	N	Y	N	N	N
Bobax	N	N	Y	Y	N	N	Y	Y	N
Ozdok	N	N	Y	Y	N	N	Y	Y	N
Virut	Y	N	Y	Y	N	N	N	Y	N
Waledac	N	N	Y	Y	N	N	Y	Y	N
Naz	N	N	Y	Y	N	Y	Y	Y	Y
Naz+	N	N	N	N	N	Y	Y	Y	Y

Table 5. Client-Side detection results ("IE" stands for "Internet Explorer")

Second, all applications but Naz and Naz+ possessed the dubious network traffic attribute; RSS applications did not download suspicious files, but all of the bots did. All bots but Virut read inordinate amounts of encoded text; legitimate RSS reader applications generally did not. Additionally, only Naz and Naz+ communicated nearly exclusively with social networking sites (benign processes read from them only a fraction of the time, and traditional bots made no such communication requests). Third, all benign applications did not exhibit the unreliable provenance attribute; none attempted to replicate itself or inject code into another process, and they all possessed a verifiable digital signature. On the other hand, all bots tested displayed this attribute, as they all copied themselves or injected code into other processes (except Naz+), and lacked a verifiable digital signature.

Of note is that no social network-based bot was misclassified as a benign process and no benign application or traditional bot was misclassified as a social network-based bot. We reiterate that our goal is to detect social network-based botnet C&C, which explains why the other bots Bobax, Ozdok, Virut and Waledac were not classified as social network-based bots. These bots may be detected by using complementary host-centric or network-centric approaches, or by applying the relaxed dubious network traffic attribute described above. This also justifies why the countermeasures presented in the paper need be integrated into a comprehensive defense framework.

Limitations. A limitation of our effectiveness analysis is the lack of real-time analysis of other social network-based botnet C&C, due to other botnets undiscovered in the wild. Moreover, our analysis is conducted in post-analysis. We plan to develop an implementation of this technique to provide real-time data gathering and evaluation. Another limitation is that any one sensor can be defeated if the malware author knows the concrete details of its implementation;

knowledge of the high-level detection model is not enough. For example, if we only seek `.exe` and compressed files, a malicious file can purposely be renamed to `.jpg` or `.html` which would bypass our file download sensor. In this case, the bot's internal logic will have extra overhead, possibly checking every file in the network traffic with these file extension to identify the malicious one, thus making this approach infeasible. Malware in general are known to have attributes that trigger many of these sensors [32] and thus it is unlikely that a bot process will effectively work and bypass all our sensors.

6.3 Performance Analysis

In order to measure the client-side countermeasure's performance, we used PassMark's PerformanceTest 7.0 benchmarking to gather information on its CPU usage [27]. We benchmarked the baseline case with no data collection running, and then ran the data collector tracking zero to five processes after an hour of data collection had passed. Running the data collector added a 4.8% overhead to the overall system, and running it to track one to five processes had between a 13.3% and 28.9% overhead (See also Figure 5). With optimization, these numbers could be lowered further.

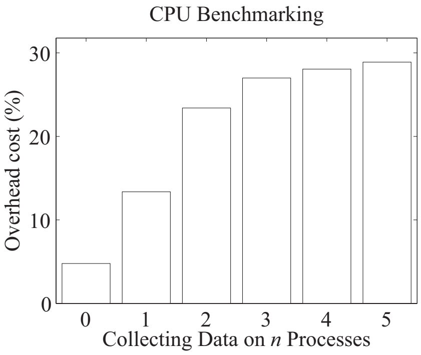


Fig. 5. CPU utilization for our client-side countermeasure process tracker

7 Integrating Server- and Client-side Countermeasures

Integrated solution. As discussed above, the server-side and client-side countermeasures are integral parts of our client/server solution to detecting social network-based botnet C&C. When each type of countermeasures deployed alone, they can be defeated in certain ways. Integrating our server-side classifier into a social network webserver is straightforward whether it operates as a library or service, and integrating the client-side detection algorithms into existing malware detection schemes or operating its sensors as its own detection framework is equally as uncomplicated. Because both systems are stand-alone, there is no need to have the systems interoperate. Indeed, a user that has the client-side solution installed while using a social network that employs the server-side solution gains the benefits of both.

Limitations. Even when our classifier is utilized by a social network provider and a machine has our client solution installed, using both still has some limitations. Specifically, if the botmaster employs steganography into their social network-based C&C, the server-side solution in its current form will not detect that message being passed. Employing steganography in such a way will diffuse the content in the message, essentially expanding the text [4]. In this case, if using popular social networks like `Twitter.com` or `FaceBook.com` with character length limitations, spreading a command over multiple messages would likely be required. More specifically, our approach can benefit from host-centric and network-centric approaches as follows.

- **A bot that reads steganographic commands and can evade our client-side sensors**. One way for a bot to evade the client-side sensors is to exist at the kernel level. Since some of the client-side solution sensors exist at the user-level, the bot can effectively bypass enough of these sensors to mask its presence on the machine. Additionally, a bot that with intimate knowledge of the *implementation details* of the client-side sensors can maneuver around our countermeasures, such as writing code that falsifies sensor data. A host-centric approach to capture additional anomalous information at the kernel level would help mitigate this attack.
- **A bot that reads steganographic commands and masquerades as a benign process**. A bot that behaves as a benign process would have to lack the self-concealing or unreliable provenance attribute. By masquerading as a benign application, say by presenting itself as a graphical application that masks its true purpose, a bot could exist with such an interface. This bot would additionally have to trick the user into starting it and keeping it running, which might prove difficult. To avoid possessing an unreliable provenance, this bot would have to have a digital signature, which is difficult to forge. Additionally, it must not dynamically inject code into another source or replicate itself, which are hallmark signs of bots, since they wish to inculcate themselves into the host machine. A network-centric solution is necessary to analyze network layer data for similar events occurring from many machines in a network during a small timeframe.
- **A bot that reads steganographic commands and runs scripts**. A bot that behaves as a social network-based bot but downloads text files instead of executables will not be classified as a social network-based bot, although it would be marked as a possibly suspicious bot. If the bot contains or is aware of a scripting engine such as a Python interpreter, the bot can run the script instead of an executable. A host-centric approach to contain general purpose malware or prevent or alert the user of script/program execution would help stop this attack. Additionally, a network-centric strategy to detect script

file downloads would help prevent the scripts from being downloaded to the client machine.

8 Conclusion and Future Work

We systematically studied a social network-based botnet and its C&C and discussed their future evolutions. We investigated, prototyped, and analyzed both server-side and client-side countermeasures, which are integral parts of a solution to the emerging threat of social network-based botnets. We also discussed how our solution can benefit from host-centric and network-centric botnet detection solutions so as to formulate a comprehensive defense against botnets.

Our future work includes: (1) implementing the client-side countermeasures as real-time detection systems, (2) improving the server-side classifier to detect steganography, (3) handling multiple stepping stones in payload redirection, and (4) and porting the client-side countermeasures to other platforms.

Acknowledgments. This work is partially supported by grants from AFOSR, ONR, AFOSR MURI, and the State of Texas Emerging Technology Fund.

References

1. E. Athanasopoulos, A. Makridakis, S. Antonatos, D. Antoniades, S. Ioannidis, K. Anagnostakis, and E. Markatos. Antisocial networks: Turning a social network into a botnet. In *Proc. Information Security Conference (ISC'08)*, pp 146–160.
2. J. Balatzar, J. Costoya, and R. Flores. The real face of koobface: The largest web 2.0 botnet explained. Technical report, Trend Micro, 2009.
3. J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *Proc. Reducing Unwanted Traffic on the Internet (SRUTI'06).*
4. M. Chapman and G. I. Davida. Plausible deniability using automated linguistic stegonagraphy. In *Conference on Infrastructure Security*, October 2002.
5. A. Cheng and M. Evans. Inside twitter: An in-depth look inside the twitter world. `http://www.sysomos.com/insidetwitter`.
6. M. Collins and M. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *Proc. RAID'07*, pp 276–295.
7. M. Collins, T. Shimeall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane. Using uncleanliness to predict future botnet addresses. In *Porc. IMC'07.*
8. E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: understanding, detecting, and disrupting botnets. In *Proc. SRUTI'05.*
9. Microsoft Corporation. Network monitor 3.3. `http://go.microsoft.com/fwlink/?LinkID=103158\&clcid=0x409`.
10. CWSandbox.org. Cwsandbox—behavior-based malware analysis. `http://www.cwsandbox.org`.
11. D. Dagon, G. Gu, C. Lee, and W. Lee. A taxonomy of botnet structures. In *Proc. ACSAC'07.*
12. DigiNinja. Kreiosc2: Poc using twitter as its command and control channel. `http://www.digininja.org`.
13. T. Easton and K. Johnson. Social zombies. In *DEFCON'09.*

14. J. Goebel and T. Holz. Rishi: identify bot contaminated hosts by irc nickname evaluation. In *Proc. HotBots'07*.
15. J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: overview and case study. In *Proc. HotBots'07*.
16. G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Security'08*.
17. G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Security'07*.
18. G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proc. NDSS'08*.
19. T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. *LEET'08*.
20. X. Hu, M. Knysz, and K. G. Shin. Rb-seeker: Auto-detection of redirection botnets. *Proc. NDSS'09*.
21. Finjan Software Inc. Web security trends report q4 2007. Technical report, Finjan Software Inc., 2007. `http://www.finjan.com/Content.aspx?id=827`.
22. J. John, A. Moshchuk, S. Gribble, and A. Krishnamurthy. Studying spamming botnets using botlab. In *Proc. NSDI'09*.
23. A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proc. HotBots'07*.
24. J. A. Morales, P. J. Clarke, Y. Deng, and B. G. Kibria. Identification of file infecting viruses through detection of self-reference replication. *Journal in Computer Virology*, 2008.
25. J. Nazario. Twitter based botnet command and control. `http://asert.arbornetworks.com/2009/08/twitter-based-botnet-command-channel`, 2009.
26. J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. *Proc. MALWARE'08*.
27. PassMark.com. Passmark performancetest 7.0. `http://www.passmark.com/products/pt.htm`.
28. S. Poland. How to create a twitter bot, 2007. `http://blog.stevepoland.com/how-to-create-a-twitter-bot/`.
29. M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proc. IMC '06*.
30. K. Singh, A. Srivastava, J. Giffin, and W. Lee. Evaluating emails feasibility for botnet command and control. In *In Proc. DSN*.
31. E. Stinson and J. C. Mitchell. Characterizing bots' remote control behavior. In *Proc. DIMVA'07*.
32. P. Szor. *The Art of Computer Virus Research and Defense*. Symantec Press, 2005.
33. Weka 3 data mining software. `http://www.cs.waikato.ac.nz/ml/weka/`.
34. Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming botnets: signatures and characteristics. In *Proc. SIGCOMM'08*, pp 171–182.
35. Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. Botgraph: large scale spamming botnet detection. In *Proc. NSDI'09*.
36. Z. Zhu, V. Yegneswaran, and Y. Chen. Using failure information analysis to detect enterprise zombies. In *Proc. Securecomm'09*.
37. L. Zhuang, J. Dunagan, D. Simon, H. Wang, I. Osipkov, G. Hulten, and J. Tygar. Characterizing botnets from email spam records. In *Proc. LEET'08*.

Analyzing and Exploiting Network Behaviors of Malware

Jose Andre Morales[1], Areej Al-Bataineh[2], Shouhuai Xu[1,2], and Ravi Sandhu[1]

[1] Institute for Cyber Security, University of Texas at San Antonio
{jose.morales, ravi.sandhu}@utsa.edu
[2] Department of Computer Science, University of Texas at San Antonio
{aalbata,shxu}@cs.utsa.edu

Abstract. In this paper we address the following questions: From a networking perspective, do malicious programs (malware, bots, viruses, etc...) behave differently from benign programs that run daily for various needs? If so, how may we exploit the differences in network behavior to detect them? To address these questions, we are systematically analyzing the behavior of a large set (at the magnitude of 2,000) of malware samples. We present our initial results after analyzing 1000 malware samples. The results show that malicious and benign programs behave quite differently from a network perspective. We are still in the process of attempting to interpret the differences, which nevertheless have been utilized to detect 31 malware samples which were not detected by any antivirus software on Virustotal.com as of 01 April 2010, giving evidence that the differences between malicious and benign network behavior has a possible use in helping stop zero-day attacks on a host machine.

1 Introduction

The ever growing sophistication of malware, especially zero-day attacks, with faster distribution and stealthier execution has forced signature based detection in an uphill battle that is difficult to win. Behavior based detection is increasingly being used by commercial software vendors with some success but is partially reliant on understanding the behavior of known malware to attempt detecting future attacks.

This research analyzes known malicious and benign samples in an attempt to exploit differences in their network behavior to accomplish accurate behavior based malware detection. The data set consisted of 1000 malware samples, including 31 not detected by any antivirus software on Virustotal.com on 01 April 2010 and 123 benign samples. The analyzed data included DNS, NetBIOS, TCP, UDP, ICMP and other network traffic. For each analyzed malware and benign sample, we collected occurrence amounts of basic network functions such as total number of DNS queries and NetBIOS query requests. Observations of captured network activity and occurrence amounts were analyzed and correlated to identify network behaviors occurring mostly in malware. We use clustering and classification algorithms to evaluate how effectively our observed network behaviors can differentiate malware from benign samples.

Given our observed network behaviors, our clustering and classification produced minimal false positives and false negatives. In addition, 31 malware samples not identified by any antivirus software on Virustotal.com on 01 April 2010 were correctly clustered and classified using our observed network behaviors. These results give evidence that the observed differences between malicious and benign network behavior can be useful in stopping zero-day attacks on a host machine.

The principal contributions of this research are:

1. Identification of network behaviors occurring mostly in malware usable in behavior based malware detection.
2. Discovery of novel malicious uses of network services by malware.
3. Evaluating the effectiveness of observed network behaviors in identifying malware and benign processes with clustering and classification.

This research presents early results of one perspective of an ongoing project dealing with malware behavior based on a sample size of 1000 malware and 41 benign processes. The benign processes were executed three times each for a total of 123 instances which were used as samples for our analysis. The goal of this ongoing research is a real time behavior based malware detection system incorporating several perspectives capable of detecting known and unknown malware on host machines.

The rest of this paper is organized as follows: Section 2 gives related work, Section 3 describes our data set, Section 4 presents our network behaviors, Section 5 gives our clustering and classification results, Section 6 discusses our approach and results, Section 7 gives limitations and Section 8 is conclusions and future work.

2 Related Work

The research of Bayer et. al. [3] presents a dynamic malware analysis platform called Anubis which is used to collect behaviors of known malware samples in a controlled environment. This system inputs a binary executable and records API invocation, network activity, and data flows. The results are used to report observed behaviors principally on the file system, registry, and network activity. The reported network activity only provides data on usage of protocol traffic, connections to remote servers, file downloads, port scanning and other typical network tasks. The results give direction as to which forms of network activity should be monitored closely for malicious events. Using Bayer et. al. as motivation, we analyzed and produced occurrence amounts of basic network functions which were used to aid in defining our set of network behaviors.

Malware binary analysis platforms such as Anubis [1], Malheur [13], Bitblaze [4] and CwSandbox [24] are designed primarily to run known malware samples in a controlled environment and record execution behavior. Recording is done via various techniques from API hooking to monitoring modifications and data flows in various OS components. These platforms record general network activity behavior which are reported to the user. The reports do not include sufficient detailed information to identify malware's precise implementation and use of network services making it difficult to discover novel malicious acts captured in network activity. Our research fills this gap by capturing finely grained network behavior facilitating detailed analysis which was key in our discovery of novel network behaviors that successfully detected several malware samples.

The research presented by Morales et. al. [15] analyzes a specific form of network activity behavior called RD-behavior which is based on a combination of DNS activity and TCP connection attempts. The authors found bot processes often use reverse DNS queries (rDNS) possibly to harvest new domain names. The rDNS often fails and is then followed by a TCP connection attempt to the input IP address of the failed rDNS, the authors regard this as an anomalous behavior. This anomalous behavior is successfully used by the authors to detect bots and non-bot malware. The approach in [15] was limited when the authors removed one of their defined behavior: a failed connection attempt to the returned IP address of a successful DNS query. Our results revealed an almost total absence of rDNS usage and several instances where malware used the removed behavior. Using this behavior helped raise our detection accuracy.

The research of Zhu et. al. [28] detected bots with a host-based technique based on high number of failed connection attempts. Measuring the connection failure rate of bots and benign processes showed that successful bot detection is achievable using only this metric. Measuring failed connection attempts may only be effective with bots that are totally or partially inactive while fully active up to date bots and other malware with little or no failed connection attempts may go undetected by this approach. Our research relates failed connection attempts with DNS, NetBIOS and other network behaviors creating a more robust approach to malware analysis and detection.

A broad corpus of research exists analyzing and detecting malware samples, families and categories [8, 17, 11, 9, 22, 6, 14, 12, 16, 18, 7, 2, 23]. All use different perspectives to measure, analyze and detect malware using host-based, network-based and hybrid approaches. Our research enhances the current literature by relating different specific network activities together to define network behaviors mostly used by malware.

3 Data Set Analysis

Our analysis is based on 1000 known malware samples and 41 benign samples. The benign samples were executed three times each for a total of 123 instances which were used as samples for our analysis. The malware samples were acquired by downloading the first 969 samples from the CWSandbox sample feed on 27 October 2009 [24]. The upload date was arbitrarily chosen. The set contains a broad range of malware types including: bots, backdoors, malware downloaders, keyloggers, password stealers and spyware amongst others, Table 1 lists prominent malware in the data set. Uploading the MD5 sums to Virustotal.com provided malware names from Kaspersky, McAfee and Symantec. We also downloaded 31 malware samples from the 31 March 2010 upload on CWSandbox malware repository. These 31 were chosen because their MD5 sums, listed in Table 2, were reported as undetected by all antivirus software used by Virustotal.com on 01 April 2010 and we were capable of executing and capturing their network behavior in our testing environment. The majority of our malware samples had successful network activity during the collection period connecting with remote hosts and conducting malicious deeds.

Prominent malware samples in data set			
Downloaders	**Bots**	**Worms**	**Hybrids**
Bifrose.bmzp	Koobface.d	Iksmas.bqs	Krap.n
PcClient.ahqy	Padobot.m	Mydoom.m	PolyCrypt.b
Poison.pg	Virut.by	Allaple.a	Refroso.qj
Turkojan.il	Zbot.acnd	Bacteraloh.h	Scar.hez
Genome.cehu	Buzus.amsz	Palevo.ddm	
CodecPack.ill			
Lipler.fhm			
Adware	**Scareware**	**Rootkits**	**Viruses**
FenomenGame	SystemSecurity.cc	Tdss.f	Sality.aa
BHO.nby	XpPoliceAV.apd		
Monderd.gen			
Benign samples in data set			
Adobe Reader	Ares	Avant	BitTorrent
Chrome	CuteFtp	DeskTube	Facebook Desktop
FileZilla	FireFox	FlickRoom	Flock
Google Talk	Google Update	IE explorer	Kaspersky Security
K-Meleon	LimeWire	Ping	PPLive
PPStream	RSSBandit	Skype	Snarfer
Snitter	SopCast	Spyware Dr.	Stream Torrent
Streamer radio	TortoiseSVN	Traceroute	TVants
Tvkoo	TVUPlayer	TweetDeck	Twhirl
uTorrent	UUSee	Win Player	Win Update
Zultrax			

Table 1. Prominent malware and benign samples in data set.

The benign test set, also listed in Table 1, covered a wide range of popular and daily used network active applications including: web browsers, FTP clients, RSS readers, social network clients, antivirus software, Peer-to-Peer (P2P) clients and standard network tools amongst others. We captured network activity in VMWare Workstation with Windows XP SP2 using Windows Network Monitor along with proprietary network layer monitors to record the network activity for an execution period of 10 minutes for each data set sample. The individual samples were manually executed one at a time in VMWare Workstation with our monitors collecting all network traffic and the captured data was saved to a local repository for analysis. The benign processes were installed, used under normal conditions and updated (when available) during testing.

The network activity of the group of 969 malware samples was collected between 27 October 2009 and 01 November 2009, the network activity of the group of 31 malware samples was collected on 01 April 2010, and the network activity of the group of 41 benign samples was collected between 01 April 2010 and 03 April 2010. Collecting network behavior of the malware samples was done immediately after downloading

31 malware not detected on Virustotal.com - 01 April 2010	
732e014e309ffab8ed9a05198d060a0b	ce1cd380910e28092f880643ec1f809d
94004413140e2022c0880f3828a1c0ee	cbed573de18b900cd91cc9e4558fb645
bcebf381a36099f697d2e00f3ec4f26e	7a84fd3ff0aa487ae2142e7130c78d9f
2fbea182c4c7d47419b2c25d72eb64bc	6d25e4a5db130cda772e09d458afacad
8a98176d289e099ccf359aaed06daf9e	bdd7bd56d65471b594c0822dd434a84f
037629b54b5714457ff2abefdab0c349	6b24b3779730f4add8d562daa1bc0ddf
7407c24f17d7c582901623c410ab7a91	8189e6f967b612e5ee7a74981278de4a
36a256686620fa7d3b9433af19cf57a2	5cfb57eac56c8639329d9ecab7b7f4ac
cde17b3c02d6143a9c1fa22eedad65ac	fbc377f7010b6a3216f7fd330dcfe69e
2e3108689a758c629286ef552e89b858	0b15d6658f306cfea3fe20bd32c91a0d
ae7d5ad001c26bbda2f32610f28484b9	9207e79e1f2191d3d44343482ab58a4e
25181c8ed97357b52ea775bc5dca353c	2bbb004cc926a071bda327ca83bf03fb
b0c89519569ce2e310958af0e5932ed1	e73da6feae4fabd251bb19f39c1a36d3
d2ebbc7609672d46e7bb8b233af585aa	e38c4a027b5a570eae8c57de8e26fcbb
bc8aa3e072fbec4045bf94375ac53be9	018197ab7020625864e6f4ff65611fc7
5dae2c8bf87e6a9ad44e52d38ac3411e	

Table 2. MD5 sums of data set malware samples not detected on VirusTotal.com

the samples to assure the samples were still active, meaning the malware would still connect with remote hosts and conduct malicious deeds producing network traffic. The vast majority of our malware samples, over 95%, produced network traffic which was the basis of our analysis.

4 Network Behavior

This research analyzes known malware and benign samples in an attempt to exploit differences in their network behavior to accomplish accurate behavior based malware detection. Differences in network behavior were identified through manual post analysis of collected network traffic. The captured network activity of our data set contained typical protocols such as TCP, UDP, and DNS but they were not always used in the normal expected way, most notably in our malware samples. We were able to collect occurrence totals of basic network functions and correlate together different occurrence amounts of specific network activity to identify network behaviors which, according to our results, occurred more often in malware than benign samples. The identified network behaviors, defined as B_n where n is an identification number, are described below.

4.1 DNS and NetBIOS

The Domain Name System (DNS) and Network Basic Input/Output System (NetBIOS) provide services to acquire IP addresses when a domain name is provided and vice versa [5, 19]. Coarse-grain occurrence amounts of both protocols by known malware has been previously shown [3, 15]. Table 3 summarizes our occurrence amounts for DNS queries, reverse DNS queries and NetBIOS name requests. The analysis revealed 100% of benign processes and 77% malware issuing DNS queries mostly due to malware's use of other network services, such as NetBIOS and ICMP, to acquire IP addresses for connection attempts. The benign samples with failed DNS queries were web browsers unable to reach third party content and P2P video and audio streamers unable to locate remote hosts for a specific stream. Several malware samples had failed DNS queries, most were domain names of malware servers that were either not active or previously discovered and shut down. Reverse DNS queries (rDNS) were notably absent with only 2% of malware and no benign samples. This contradicts the findings of [15] which documented bots and non-bot malware performing rDNS and conjectured these queries were an essential component to establish malicious network activity. It can be inferred, from testing our samples, that the current generation of malware may possibly be less reliant on rDNS in favor of other techniques providing the same IP address and domain name related information.

Analyzing the occurrence totals of NetBIOS name requests (NBTNS) revealed 56% of malware and 4% of benign samples implemented this activity. The benign samples with NetBIOS name requests were the web browsers Google Chrome with fifteen name requests and Firefox with six name requests. Further analysis revealed the domain names used in the NetBIOS name request of Google Chrome and Firefox had first been used in a DNS query with some failing and others succeeding. The malware samples revealed two distinct forms of NetBIOS name request usage: (i) expected usage, same as benign, and (ii) performing NetBIOS name requests on domain names that were not part of a captured DNS or rDNS query. To our knowledge, the second form is a novel observation of NetBIOS use by malware not presented in previous research. Of the 1000 malware samples, 49% exhibited the second NetBIOS usage described here. We concluded this was a network behavior occurring mostly in malware and usable for detection. Based on this, we define the following network behavior:

- B_1: A process performs a NetBIOS name request on a domain name that is not part of a DNS or rDNS query.

Table 3 shows B_1 occurring only in malware, with 49%. Using online malware databases such as MalwareURL.com, we found many domain names used by our malware samples in B_1 identified as malware servers, but several other domains did not show up leading us to believe they were recently created and registered, inactive, had avoided detection, were infected hosts, or newly activated servers. We conjecture malware uses behavior B_1 in an attempt to acquire remote host information while avoiding detection by anti-malware that may not monitor NetBIOS but most probably does monitor DNS.

4.2 RD-behavior

This network behavior as originally defined [15] was primarily based on frequent usage of reverse DNS queries (rDNS) by bots. The authors defined four network behavior paths of which three included rDNS. Their results implied rDNS combined with TCP connection attempts was sufficient to detect malware and eliminated false positives by

Samples with	**Malware 1000 samples**	**Benign 123 samples**
DNS queries	77%	100%
Reverse DNS queries	2%	0%
NetBIOS name requests	56%	4%
Behavior B_1	49%	0%

Table 3. Samples with DNS, NetBIOS, & B_1

omitting the only behavior path dealing solely with DNS queries. Our analysis revealed a notable absence of rDNS and a high occurrence of DNS queries, see Table 3, many of which exhibited the omitted behavior. We conjecture better detection can be achieved by including all four behaviors from [15] redefined as follows:

- B_2: Failed connection attempt to an IP address obtained from a successful DNS query.
- B_3: Failed connection attempt to the input IP address of a successful rDNS query.
- B_4: Connection attempt to the input IP address of a failed rDNS query.

In [15] behavior path P_5 is defined as: A successful connection to an IP address used in a failed rDNS query and behavior path P_6 is defined as: A failure to connect with an IP address used in a failed rDNS query. We reduced the number of network behaviors by combining behavior paths P_5 and P_6 into one network behavior B_4. Behavior B_2 implies a successful connection should occur to IP addresses obtained in successful DNS queries, a failed connection attempt indicates something is not right and should be investigated. Malware can exhibit this behavior when domain names have been shut down or taken offline and their DNS records have not been updated or removed. Behavior B_3 has the same implication as B_2 but with the input IP address of rDNS queries. Behavior B_4 is assumed to only occur in malware. We assuem an input IP address failing an rDNS query as unreachable and should not be used for connection attempts. Table 4 shows total number of processes with behaviors B_2, B_3 and B_4. Our occurrence amounts showed 21% of malware and no benign samples with B_2 and no occurrences of B_3 and B_4 due to very low rDNS usage. These results imply rDNS may be used less often by malware in favor of other techniques providing the same information in a more clandestine manner.

4.3 UDP and ICMP

Traffic between local and remote hosts using captured User Datagram Protocol (UDP) [25] did not serve a significant role, except for DNS and rDNS, in our analysis due to similar occurrence amounts of network activity in both malware and benign. Previous research [3] has documented coarse-grain UDP occurrence amounts by malware, but does not include a comparison with benign processes. Identifying network activity behaviors in the UDP protocol is part of our ongoing research.

Samples with	**Malware 1000 samples**	**Benign 123 samples**
Behavior B_2	21%	0%
Behavior B_3	0%	0%
Behavior B_4	0%	0%

Table 4. Samples with behaviors B_2, B_3 & B_4

The occurrence amounts of Internet Control Message Protocol (ICMP) [10] activity, which focused on ICMP echo requests and replies, revealed an elevated usage by the malware samples in comparison to the benign samples. Further analysis concluded that malware was using ICMP echo requests in the same manner as the Ping network utility [20] to decide if a remote host was reachable, thus being a candidate for a connection attempt. Malware use of ICMP has been previously observed [27] but was not distinguished as a behavior frequently used by malware in comparison to benign. Our analysis showed malware never attempted connections to IP addresses not receiving a reply to an ICMP echo request and almost always attempted to connect with IP addresses that did have a successful reply. Furthermore, the input IP address of the echo requests were never part of a DNS or rDNS query or NetBIOS name request leading to conclude these IP addresses were hardwired, dynamically generated, or downloaded from a malware server. Based on these observations, we define two network behaviors as follows:

- B_5: ICMP only activity, ICMP echo requests for a specific non-local network IP address with no reply or a returned error message.
- B_6: TCP/ICMP activity, TCP connection attempts to non-local IP addresses that received a successful reply to their ICMP echo requests.

We assume the IP addresses used in B_5 and B_6 are never part of DNS, rDNS or NetBIOS activity. This assumption is supported by our observations of the captured network activity. The results of this analysis are listed in Table 5. B_5 occurred more often in benign than malware but the benign samples also used ICMP less than malware, perhaps favoring other similar and more conventional services such as DNS queries, see Table 3. B_6 was exhibited in 11% of malware and only 2% benign samples. This supports our claim that malware frequents ICMP use to identify IP addresses for connection attempts. Our observations of B_5 and B_6 are, to our knowledge, novel in the literature not being previously reported.

4.4 Other network activity

This encapsulates other less occurring activities which were considered significant since they rarely occurred in any of our data set samples or were implemented in a non-conventional way. We consider these network activities to be anomalous and not necessarily malicious behaviors. The value of recording occurrences of these behaviors is in cases where a novel and never before observed, or rarely used malicious behavior occurs in a malware sample. We encompass this idea with the following behavior:

Samples with	**Malware 1000 samples**	**Benign 123 samples**
Behavior B_5	3%	4%
Behavior B_6	11%	2%

Table 5. Samples with behaviors B_5 & B_6

Samples with	**Malware 1000 samples**	**Benign 123 samples**
TCP connection attempts to IP addresses never used in DNS, NetBIOS, ICMP	10%	2%
Listen connections on non-typical port numbers	2%	7%
Successful DNS queries returning local network IP addresses	1%	0%
Use of non-typical network protocols and commands	4%	0%
Behavior B_7	18%	9%

Table 6. Samples with behavior B_7

- B_7: Network activity that is rarely occurring or implemented in an anomalous manner.

Table 6 lists the amount of samples exhibiting the different types of observed network activity and B_7. TCP connection attempts to IP addresses which were not part of DNS, NetBIOS or ICMP activity were the most prominent in this group with 10% in malware and only 2% in benign. These malware, upon initial execution, immediately attempted connections to IP addresses ranging from a few to over one hundred different addresses which appeared to have been hardwired or dynamically generated. The benign sample with this activity was the video chat program Skype which connected to a server during installation.

Second most prevalent network activity was use of non-typical protocols and network commands with 4% in malware none in benign. The malware attempted connections using either FTP or SMB or RTCP. These were the only samples from our data set using these protocols except for FTP which is a typical protocol; the reason we documented FTP usage is the malware had a very small amount of FTP activity download from a remote server along with a much lager amount of TCP and UDP traffic.

One malware sample used the authentication system KerebosV5 and one other malware sample used the network command suite Andx. Interestingly, the Andx commands were attempting to authenticate and access local network IP addresses in search of a file server perhaps to host inappropriate content. Listening TCP connections using non-typical port numbers occurred in 2% malware and 7% benign samples. Malware listened on non-typical or private ports [21] such as port numbers: 19178, 24450, 25254, 27145 and 36975; benign also listened on non-typical or private ports such as port numbers:

19396, 33680, 36363 and 58480. Two malware samples performed successful DNS queries on domain names returning local network IP addresses: gogog029.100webspace.net - 127.0.0.1 and probooter2009.no-ip.org - 0.0.0.0. It is unclear if these DNS query results were modified by the malware or if these were intentionally returned by the DNS server. B_7 was exhibited in 18% malware and 9% benign, suggesting rarely or anomalous occuring network activity may be useful in differentiating malware and benign.

5 Clustering and Classification

To evaluate how effectively our observed network behaviors can differentiate between malicious and benign samples, we input the data through clustering and classification algorithms using the Weka data mining software [26]. Clustering and classification algorithms are extensively used in the literature to evaluate proposed host, network and hybrid detection approaches and are well established as accurate indicators of effectiveness and efficiency of a proposed detection approach. Our data set consisted of the occurrence amounts of network behaviors B_1 through B_7, discussed in Section 4, for each malware and benign sample. The complete data set was used for clustering; for classification, the training set contained the first 700 malware samples and 40 benign with the test set containing the remaining samples. The 31 undetected malware samples were not part of the training set. Some of the samples in the test set not found in the training set are listed in Table 7.

Malware samples	**Benign samples**
BHO.nby	Adobe Reader
Mabezat.b	BitTorrent
Monderd.gen	Chrome
Poison.pg	CuteFtp
Swizzor.a (2)	Facebook Desktop
Turkojan.il	FlickRoom
VB.bfo	Kaspersky Security
VB.vr	Skype
31 undetected malware	SopCast
	TVants

Table 7. Some of the malware and benign samples in test set and not in training set

5.1 Clustering results

The data set was input to the complete suite of clustering algorithms in Weka. The top three results are listed in Table 8. False positives and false negatives were determined by observing if the majority of a cluster was composed of malware or benign samples. If malware was the majority then the benign samples were classified as false positives; if benign was the majority then the malware samples were classified as false negatives.

Clustering algorithm	**Number of clusters**	**True positives**	**True negatives**	**False positives**	**False negatives**	**FP rate**	**FN rate**
DBScan	8	119	1000	4	0	0.4%	0%
Expectation maximization (EM)	4	123	988	0	12	0%	1%
Xmeans	3	123	1000	0	0	0%	0%

Table 8. Top three clustering results with 1000 malware and 123 benign samples

DBScan and EM algorithms produced encouraging results with no false negatives in the first and no false positives in the second algorithm. The four false positives produced by DBScan were SopCast, TVUPlayer, UUsee media center, and TVants. All of these are video streamers whose content source comes from several IP addresses which are constantly changed and removed, making it difficult to keep up to date. This is very similar to IP addresses used by malware authors, especially in botnets [18], which constantly change primarily to avoid detection. All four were grouped in one cluster with many different classes of malware, the samples in this cluster exhibited many instances of behaviors B_1, B_2 and B_7. The main reason why the four false positives were grouped in this cluster was due to having between 3 and 8 instances of behavior B_2. In each case, we attempted to access several video streams. Many of these were unreachable and analyzing the network activity showed the failed connection attempts to IP addresses of successful DNS queries. Further investigation into these IP addresses revealed they were temporary video content servers where the specific video streams were no longer available. The IP was taken offline but the records pointing to them had not been removed from the software's database of active streamers.

The twelve false negatives produced by the EM algorithm consisted of nine malware downloaders, three of which belong to the packed.win32.krap family, one worm, one bot (koobface) and one of the 31 undetected malware samples with MD5 hash value *7407c24f17d7c582901623c410ab7a91*. Three samples: koobface and two malware downloaders were seemingly inactive having no successful connection attempts with remote hosts and only four samples exhibited at most a single instance of just one of the following behavior symptoms: B_1, B_2, B_6, B_7. The small amount of network behaviors produced by these malware led to their false negative production since their network traffic was very similar to the benign samples.

The Xmeans algorithm produced no false positives and no false negatives, with all malware grouped in two clusters and benign in one cluster. The 31 undetected malware samples, see Table 2, were correctly clustered by both Xmeans and DbScan while EM correctly clustered 30 implying our network behaviors can detect malware missed by commercial antivirus software and may be usable in stopping zero-day attacks. Overall, the clustering suggest our network behaviors are capable of detecting malware with minimal false positives and false negatives.

5.2 Classification results

Several classification algorithms were applied on the test set with BayesNet, NNge, Random Forest and Rotation Forest producing the best results listed in Table 9. The false negative rates for all four algorithms were low ranging from 0.6% to 1%, the false positives were also very low ranging between 0% to 2%. All the algorithms had the same two malware samples, VB.vr and one of the 31 undetected malware (MD5 hash value $25181c8ed97357b52ea775bc5dca353c$) as false negatives. Both of these malware were not part of the training set, exhibited 3 or less instances of behavior B_5 with different IP addresses and had successful network activity with remote hosts whose IP addresses were acquired through successful DNS queries. The third false negative produced by BayesNet was one of the 31 undetected malware (MD5 hash value $cbed573de18b900cd91cc9e4558fb645$) which was active, had two instances of behavior B_5 on two different IP addresses and was not in the training set.

TVants and SopCast were the only two processes flagged as false positives. These two samples were also clustered as false positives. The reason was again their failed connection attempts to IP addresses which were no longer online hosting a video stream thus producing instances of behavior B_2. Only one of the 31 undetected malware was flagged as false negative by all four of our algorithms, with one more being flagged by BayesNet. The other 29 undetected malware were all correctly classified by all four algorithms. This result further confirms the capability of our behaviors and occurrence amounts to detect malware not detected by commercial antivirus software and gives further evidence to their use in helping stop zero-day attacks. Overall, the classification results further suggest our network behaviors can correctly classify both known and unknown malware.

Classification algorithm	**False positives**	**False negatives**	**FP rate**	**FN rate**
BayesNet	1	3	1%	1%
NNge	1	2	1%	0.6%
Random forest	0	2	0%	0.6%
Rotation forest	2	2	2%	0.6%

Table 9. Top four classification test set results with 300 malware and 83 benign samples

6 Discussion

According to our results in Section 4, of the seven defined behaviors, B_1 occurred the most in the malware samples with 49% followed by B_2 with 21% and B_7 with 18%. All three are considered behaviors more likely to occur in malware than in benign processes with B_7 initially assumed anomalous and not necessarily malicious. Behaviors B_1, B_5 and B_6 are, to our knowledge, novel observations implemented by malware to locate active remote hosts for connection attempts and, in our tests, occurred more in malware than benign. Behavior B_7 is particularly interesting due to its subjective nature which can encapsulate any network activity considered significant and rarely occurring. Therefore it is easy to add activities which degrade detection accuracy. A knowledge expert is best suited to compose activities which comprise this behavior.

Our clustering results were better than expected with perfect results in the case of Xmeans, implying our network behaviors are capable of providing accurate malware detection. Our data set covered a wide spectrum of known malware and benign classes and was able to train our classifiers to correctly identify the majority of malware in the test set with minimal false positives and false negatives.

The most interesting aspect of the results was the highly accurate clustering and classification of the 31 undetected malware. The MD5 sums of all 31 samples were not detected by any antivirus software on Virustotal.com on 01 April 2010 yet our testing correctly identified them with minimal exceptions. This detection accuracy gives strong evidence that our behaviors can help stop zero-day attacks on a host machine, especially in cases where signature-based detectors fail to identify a zero-day attack.

A robust detection system encompasses several malware detection perspectives. This research has only studied one of these perspectives, network activity, in a behavior based way to avoid implementing a detection methodology dependent on malware signatures. Part of our ongoing research is to combine our findings of the network activity perspective with other perspectives to produce a more complete behavior based malware detection system.

7 Limitations

Several protocols such as ARP and SMB were not studied. Their value to enhance our detection accuracy is being analyzed and added to current results. All analysis was done in a virtual machine which forcibly excluded interesting malware samples that are VM aware and ceased to execute or masqueraded as benign upon VM detection. The data set consisted only of malware samples which are initially executed by a mouse double click. Malware packaged as a dll file, kernel system service, or non-executable were not used. We are developing tools allowing the execution of any malware sample regardless of its format.

8 Conclusion and Future Work

This research analyzes known malware and benign samples in an attempt to exploit differences in their network activity behavior to accomplish accurate behavior based malware detection. By analyzing and comparing known malware and benign processes, we have successfully exploited differences in their network activity behavior and produced accurate and effective malware detection with minimal false positives and false negatives. This was accomplished by producing a set of behaviors which occurred most often in our analyzed malware samples during which two novel behaviors frequently used by malware were discovered.

Our analysis results successfully clustered a diverse group of malware and benign process with very high accuracy and minimal false positives and false negatives. Classification algorithms correctly detected newly introduced malware samples also with minimal false negatives and false positives. Most interestingly, our data set included 31 malware samples whose MD5 sums were not detected by any antivirus software on Virustotal.com on 01 April 2010. These undetected malware were correctly identified using our analysis in both clustering and classification algorithms with few exceptions. This provides strong evidence that our identified behaviors can be used together with existing anti-malware solutions, especially signature-based antivirus software, to help stop zero-day attacks on a host machine. This research has presented early results on one perspective, namely network activity, of a larger ongoing project to develop a behavior based malware detection system.

Future work includes examining a suite of protocols for yet-to-be observed activity usable in creating new behaviors and refining our current behavior set and evaluation methodology to further increase detection effectiveness. Alos implementing our network behaviors in a real time detection prototype to measure the efficiency of such an approach including resource usage in collecting data in heavy traffic flows and precise measurements of elapsed time used to detect a malicious process.

Acknowledgement

This work is partially supported by grants from AFOSR, ONR, AFOSR MURI, and the State of Texas Emerging Technology Fund.

References

1. http://anubis.iseclab.org/.
2. J. Balatzar, J. Costoya, and R. Flores. The real face of koobface: The largest web 2.0 botnet explained. Technical report, Trend Micro, 2009.
3. U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel. A view on current malware behaviors. In *LEET 2009: Usenix Workshop on Large-scale Exploits and Emergent Threats*, 2009.
4. http://bitblaze.cs.berkeley.edu/.
5. http://tools.ietf.org/html/rfc1034.
6. D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia. A behavioral approach to worm detection. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 43–53, New York, NY, USA, 2004. ACM Press.
7. G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.
8. A. Gupta, P. Kuppili, A. Akella, and P. Barford. An empirical study of malware evolution. In *COMSNETS'09: Proceedings of the First international conference on COMmunication Systems And NETworks*, pages 356–365, Piscataway, NJ, USA, 2009. IEEE Press.
9. T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.

10. http://tools.ietf.org/html/rfc792.
11. X. Jiang and D. Xu. Profiling self-propagating worms via behavioral footprinting. In *WORM '06: Proceedings of the 4th ACM workshop on Recurring malcode*, pages 17–24, New York, NY, USA, 2006. ACM.
12. C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *18th Usenix Security Symposium*, 2009.
13. http://www.mlsec.org/malheur/.
14. D. Moore, C. Shannon, and k. claffy. Code-red: a case study on the spread and victims of an internet worm. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 273–284, New York, NY, USA, 2002. ACM.
15. J. A. Morales, A. Al-Bataineh, S. Xu, and R. Sandhu. Analyzing dns activities of bot processes. In *MALWARE 2009: Proceedings of the 4th International Conference on Malicious and Unwanted Software*, pages 98–103, 2009.
16. J. A. Morales, P. J. Clarke, Y. Deng, and B. G. Kibria. Identification of file infecting viruses through detection of self-reference replication. *Journal in Computer Virology Special EICAR conference invited paper issue*, 2008.
17. R. Moskovitch, Y. Elovici, and L. Rokach. Detection of unknown computer worms based on behavioral classification of the host. *Comput. Stat. Data Anal.*, 52(9):4544–4566, 2008.
18. J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. *3rd International Conference on Malicious and Unwanted Software, MALWARE 2008*, pages 24–31, 2008.
19. http://tools.ietf.org/html/rfc1001#ref-2.
20. http://en.wikipedia.org/wiki/Ping.
21. http://en.wikipedia.org/wiki/
List_of_TCP_and_UDP_port_numbers.
22. J. C. Rabek, R. I. Khazan, S. M. Lewandowski, and R. K. Cunningham. Detection of injected, dynamically generated, and obfuscated malicious code. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 76–82, New York, NY, USA, 2003. ACM Press.
23. E. Stinson and J. C. Mitchell. Characterizing bots' remote control behavior. In *DIMVA '07: Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 89–108, Berlin, Heidelberg, 2007. Springer-Verlag.
24. http://www.sunbeltsoftware.com/Malware-Research-Analysis-Tools/Sunbelt-CWSandbox/.
25. http://tools.ietf.org/html/rfc768.
26. I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.
27. H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127, New York, NY, USA, 2007. ACM.
28. Z. Zhu, V. Yegneswaran, and Y. Chen. Using failure information analysis to detect enterprise zombies. *5th International ICST Conference on Security and Privacy in Communication Networks (Securecomm 09)*, 2009.

Building Malware Infection Trees

Jose Andre Morales[1], Michael Main[2], Weiliang Luo[3], Shouhuai Xu[2,3] and Ravi Sandhu[2,3]
[1] Software Engineering Institute, Carnegie Mellon University*
[2] Institute for Cyber Security, University of Texas at San Antonio
[3] Department of Computer Science, University of Texas at San Antonio

Abstract

Dynamic analysis of malware is an ever evolving and challenging task. A malware infection tree (MiT) can assist in analysis by identifying processes and files related to a specific malware sample. In this paper we propose an abstract approach to building a comprehensive MiT based on rules describing execution events essential to malware infection strategies of files and processes. The MiT is built using strong and weak bonds between processes and files which are based on transitivity of information and creator/created relationships. The abstract approach facilitates usage on any operating system platform. We implement the rules on the Windows Vista operating system using a custom built tool named MiTCoN which was used in a small scale analysis and infection tree creation of a diverse set of 5800 known malware samples. Results analysis revealed a significant occurrent of our rules within a very short span of time. We demonstrate our rule set can effectively and efficiently build infection trees linking all related processes and files of a specific malware sample with no false positives. We also tested the possible usability of a MiT in disinfecting a system which yielded a 100% success rate.

1 Introduction

The release of never before seen malware into the wild poses a severe global threat to vulnerable systems given the difficulty to detect via signature based anti-malware programs. Possible detection can be achieved with heuristics but not guaranteed to fully eradicate the malware which leaves disinfection as the next best option. To comprehensively disinfect a system, the malware must be analyzed. The analysis must effectively and efficiently detail the infection process in a meaningful way primarily documenting the files and processes which are created or modified by the malware. Analyzing the infection process can be aided by building a malware infection tree (MiT). A MiT is a directed tree structure where each node represents a file or process and each edge represents the execution event rule causing node to join the tree. To correctly build a MiT, an understanding of the essential characteristics of malware infection is required. From the seminal definitions provided by Cohen [2] and Adleman [1], an executable file labeled a virus has the fundamental ability to self-replicate which we consider to be a basic construct for a MiT and it is essential to understand the different ways in which a virus can implement this construct on various operating systems. Previous work such as [10, 9] has shown some implementations of self-replication while others have attempted to record malware behavior using various graph structures [7, 8]. A more comprehensive MiT includes processes modified by malware. There are many known ways [12, 3] in which a process modifies other already running processes. This technique is primarily implemented via a memory injection resulting in the modified process performing anomalous, and often nefarious, events. Self-replication and memory injection create a strong bond between related processes and files and are the basic constructs of our MiT. Our MiT is further enhanced with constructs recoding the creation of files and processes by a malware not involving self-replication or memory injetion which create weak bonds. Implementing the various ways in which a malware infection can occur is highly OS dependent. It is imperative to collect needed data of a malware infection in as low a level as possible to assure building of a comprehensive MiT. Some malware, such as a rootkit [5, 13] will execute at deep or privileged OS levels hiding and avoiding detection while infecting the system. In this paper, we present an abstract approach to building MiTs using execution events rules. The rules describe execution events essential to malware infection strategies

*This research was performed in the University of Texas at San Antonio on files and processes. MiTs are built based on strong and weak bonds between relevant files and processes. We describe rule implementation in the Windows Vista operating system with our custom tool, named MiT-CoN, that builds MiTs in real time. The tool analyzed and built MiTs of over 5800 diverse known malware samples. We evaluate the efficiency of our approach by recording system stability during MiT creation and timewise analysis of MiT creation. Effectiveness was evaluated by measuring frequency of rules, timewise occurrence of rules, a comparison to infection structures of comparable systems for false positive production, and attempt system disinfection using only the MiT as a guide. Our analysis revealed our MiTs were constructed within 7 seconds of initial execution without any noticeable system instability . Our disinfection attempts yielded a 100% success rate implying MiTs may be useable in real system disinfection scenarios. The contributions of our paper are as follows:

- Propose an abstract approach to building malware infection trees (MiTs).
- Define execution event rules describing essential components of infection strategies.
- Describe implementation in the Windows Vista OS User and Kernel levels.

Several comparable systems such as Anubis, BitBlaze, JoeBox, CwSandBox, and Malheur perform dynamic analysis of a submitted sample and return a tree like structure of related files and processes. The tree edges are based on operating system specific execution events which objectively link two nodes together. The critical problem is tree creation is based on interpretation of collected execution data which can result in a tree that loosely relates files and processes which may not even be part of the malware infection such as processes and files routinely used in standard OS operations. This misinformation does not correctly represent a malware infection and can result in false positives. Our approach creates more meaningful trees by creating execution event rules based on fundamental malware infection characteristics producing fundamental bonds between nodes possibly reducing false positives. The result of our approach is a tightly bound tree structure containing minimal or no non-related data leading to a minimization of false positives and a much more realistic representation of the analyzed malware's infection strategies. The rest of this paper is organized as follows: Section 2 presents the rules used to create a MiT, Section 3 is our tool implementation, Section 4 is our analysis & results, and Section 5 is conclusions & future work.

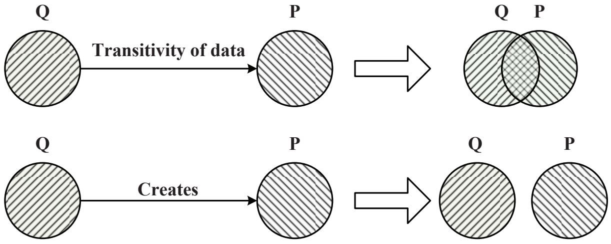


Figure 1. MiT strong & weak bonds

2 MiT Construction Rules

Strong and Weak Bonds. An essential component of our MiT building approach is linking nodes together in a strong or weak bond, illustrated in Figure 1, relationship based on the malware infection related interaction between two nodes during execution. We define a strong bond between a source node Q and destination node P when a transfer of data from Q to P occurs. The data is part of Q and transferring it to P strongly bonds both nodes based on the transitivity of data creating an intersection of identical data between Q and P. We define a weak bond when a source node Q arbitrarily creates a destination node P and there is no transitivity of data. This weak bond can be viewed as a creator/created relationship since P exists because of Q but there is no intersection of identical data between both. Other comparable systems create infection trees based only observed execution events and their nodes are not linked in a fundamentally meaningful way allowing addition of non-related nodes to the tree. The bonds between nodes of our MiTs are defined in the fundamental realm of malware infection strategies producing an infection tree with the reduced likelihood of including non-malware related nodes. A strongly bonded MiT will consist of essential files and processes directly descending from the original malware executable due to the transitivity of data between nodes and should be eradicated first in a disinfection strategy to prevent further infection and injury. Enhancing the MiT with weak bonds provides those files and processes that may not be essential to the malware's infection and injury but still should be eradicated during disinfection.

Construction Rules. We define a malware infection tree (MiT) as the output of a function $M(\cdot)$ on inputs X with a set of execution event rules R, where X is the executable file being analyzed, and R contains a set of rules $r_1 ... r_n$ that define the conditions under which an object becomes part of a MiT. $M(X)$ outputs a MiT with a directed graph as MiT $= (N, E)$, where

N is a set of nodes $n_1...n_z$ with $z \geq 2$, and E is a set of ordered pairs of nodes $(n_i, n_j) \in N^2$ that create an edge between n_i and n_j. A node n is either a file in the file system or a currently running process. We assume X is always the root node of a MiT. Once a new node n is added to a MiT, its infection execution events are recorded and used to add further nodes to the MiT. If $z = 1$, then only one node n, presumably the root node X, is present in the MiT and thus no tree was created. MiT construction is based on the rules in R. The rules are mostly based on the fundamental definition of malware, Cohen [2] & Adleman [1], which stipulate a malware, specifically viruses and worms, must replicate to be classified as such. The rules also reflect a malware's tendency to nefariously modify running processes.

File System Rules. We consider a malware can infect via self replication into the file system in two primary ways. First, using a call such as *copy*(n, m), where n is a node of a MiT and the caller of the *copy* function. In this case n creates a new file which is an exact copy of itself. Second, using a series of calls such as $read(n, q); write(q, m)$, where n is a node of a MiT and the caller of $read$ and $write$, q is some temporary storage, and m is an already existing file that is being modified by n. The modification can be achieved by n prefixing, suffixing, overwriting or randomly writing some or all of its own data into the file m. Note in both cases, n is the source of the replications and the caller of the operations, therefore n is invoking self-reference replication as described in [9]. We also consider a malware that can infect a system without requiring self replication. This can be donw through the arbitrary creation of files such as $createfile(m)$ in the file system, where an existing node n calls the operation and creates a file that contains no data originating in n. There are many known malware samples that create files during execution for several reasons such as logs, configurations or to store data later sent to a remote host. Even though these files are not created via self replication they are valid components of an infection and should be included in a MiT. Based on these considerations, we define the following three file system execution event rules for the construction of a MiT:

$f1$**:Infection via self replication.** A file m becomes a new node $n \in N$ of a MiT if and only if $P(replicate(P, m)) \rightarrow True$ where a currently running process P is a pre-existent node in a MiT and P has issued a replication request where P itself is the source parameter and m is the destination parameter. m becomes a new node in the MiT where the node for P is located and the outgoing edge (P, m) is labeled $f1$. Note that m can be a newly created file by P or an existing file and the replication from P to m can be complete or partial. $f1$ exhibits transitivity of data and is labeled a strong bond.

$f2$**:Infection via arbitrary file creation.** A file m becomes a new node $n \in N$ of a MiT if and only if $P(filecreate(m)) \rightarrow True$ where a currently running process P is a pre-existent node in a MiT and P has issued a file creation request where m, a newly created file, is the destination parameter. m becomes a new node in the MiT where the node for P is located and the outgoing edge (P, m) is labeled $f2$. $f2$ is a creator/created relationship with no transitivity of data and is labeled a weak bond.

$f3$**:Infection via arbitrary file write modification.** A file m becomes a new node $n \in N$ of a MiT if and only if $P(filewrite(m)) \rightarrow True$ where a currently running process P is a pre-existent node in a MiT and P has issued a file write request where m, a pre-existing file, is the destination parameter. m becomes a new node in the MiT where the node for P is located and the outgoing edge (P, m) is labeled $f3$. Note P may modify m either by prefixing, suffixing, overwriting or randomly writing data into m. The essential component of $f3$ is the data being written to m comes from some other location and not from P. $f3$ does not exhibit transitivity of data and is labeled a weak bond.

Process Rules. We consider malware that can infect a system via process manipulation in two primary ways. First, a malware can modify the memory range of a currently running process. This is typically done via dynamic code injection [12, 3]. A malware will write (inject) nefarious code into the allocated memory of some other process and then spawn a new process instance which executes the just injected code. This results in the victim process performing execution events it would otherwise not do under benign conditions. Several known malware inject nefarious code into system critical processes. Malware authors assume there is a high unlikelihood that a user would terminate these processes since they are considered critical to OS functionality. Second, malware will create processes from the static file images of executables that either were created or downloaded to the system. Malware is known to copy or download from remote malicious servers to the system other malware as part of its payload, in many cases a trojan(s) such as password stealer, key loggers, and spam engines. In some cases, a malware may self replicate by spawning multiple processes of itself running on a system to either overwhelm the system or survive an anti-malware detection and removal attempt. Process manipulation is a very powerful tool for malware facilitating: system compromise, increased chances of detection survival, and delegate nefarious goals to benign processes. Injection is particularly powerful allowing nefarious deeds to possibly go unnoticed when carried out by benign system critical processes. Based on these considerations, we define the following two process execution event rules for the construction of a MiT:

$p1$**:Infection via dynamic code injection of a currently running process.** The static file image m of a currently running process P becomes a new node $n \in N$ of a MiT if and only if $Q(codeinject(P,d)); Q(newprocessinstance(P)) \rightarrow True$ where Q is some currently running process and a pre-existent node of a MiT, writes data, presumably code instructions stored in Q's memory space or static file image, into the allocated memory space of P and then spawns a new instance of P which executes the just written data. m becomes a new node in the MiT where the node for Q is located and the outgoing edge (Q,m) is labeled $p1$. P is a pre-existing and presumed benign currently running process of the system which gets nefariously modified by Q in memory. The static file image m of the modified process P is never modified by Q. $p1$ exhibits transitivity of data and is labeled a strong bond.

$p2$**:Infection via process spawning.** The static file image m of a currently running process P becomes a new node $n \in N$ of a MiT if and only if $Q(newprocess(m)) \rightarrow True$ where Q, a currently running process and a pre-existent member of a MiT, invokes the command to create a new process P from a static file image m. The static file image m is also a pre-existent node of the same MiT as Q. An outgoing edge (Q,m) is created and labeled $p2$ which stores information about the static file image m and the newly created process P. In this scenario, Q spawns new processes from files that are already part of the malware's MiT including Q's own static file image. Some of these files were either created or downloaded by some node of the MiT. The key element in this rule is that m is a malware related file and part of the MiT. As opposed to $p1$ where malware injects nefarious code into benign processes then creates a new instance, $p2$ creates a process from a nefarious static file image m. The nefariousness of m is based on its pre-existing inclusion as a node of a MiT. It is possible for m to be created by a process R using the file system rules above and then spawned as a new process by Q. Both R and Q are separate nodes of the same MiT, this would produce two incoming edges to m: one for the file creation and one for the process spawning. Note that m can be the static file image of Q, meaning $p2$ also records when Q creates a new process instance of itself. In this case the outgoing edge (Q,m) contains the same static file image information for m and different identification information for the newly spawned process P. ps is a creator/created relationship with no transitivity of data and is labeled a weak bond.

Implementing the file system and process rules can produce an intersection of usage based on the target OS. For example, $f2$ and $f3$ may be invoked as subroutines of $f1$, $f3$ may be invoked as a subroutine of $f2$, and $f2$ can be invoked as a subroutine of both $f1$ and $f3$. Another example is the OS performing code injection to create both new processes and instances of already running processes. In this case, extra information about the process and its injector must be acquired to adequately decide if $p1$ or $p2$ has occurred. To allow some reasonable flexibility of the rules to accommodate unavoidable intersections we assume the execution event rules describe abstract scenarios that may invoke other rules as subroutines and explicitly recognizing the use of these other rules is not required. A MiT can have multiple edges resulting from file system and process rules occurring on the same node. When building a MiT, it is allowable to have multiple edges between nodes reflective the rule justifying the edge's existence. A possible scenario may be a process P creates a file m which then creates a file o. The file o is then spawned as a new process by P. This would create two incoming edges to o, the first edge is (m,o) labeled $f1$ and the second edge is (P,o) labeled $p2$. Having a multi-edged MiT provides richer data for disinfection. Based on the MiT a user can assess which nodes should be dealt with first, perhaps using the number of incoming and outgoing edges as a weighted determination scale. Those nodes with greater number of incoming/outgoing edges may receive higher priority over other less populated nodes. Once a MiT is created, some of the files and processes belonging to the MiT may no longer exist. Several known malware samples delete files and terminate processes which may belong to its own MiT. A typical case may be a malware X creates several descendant files some of which are spawned as new processes. Then some time later one of these descendant processes terminates X and/or erases its static file image from the file system. Another scenario is X or any other spawned processes terminating itself and/or deleting its own file from the system. These scenarios do not invalidate the MiT as the file or process did in fact exist at the moment of addition as a node to the MiT. The key challenge in implementing these rules is understanding the various ways a file or process can be created or modified in a specific OS. Once this is understood, an implementation can be created at an appropriate OS level that captures all or most of the studied implementations. We address this challenge in the next section which presents our MiT construction tool for the Windows Vista OS platform.

3 MiTCoN: Windows Vista Implementation

Our malware infection tree construction tool, named MiTCoN, is a Windows command line application which implements the execution event rules in the Windows Vista platform which outputs a table representation of a MiT. The MiT is built in real time by monitoring the samples' execution behaviors. MiTCoN takes as input the absolute path of the target WIN32 PE executable set as the MiT's root node which facilitates MiT building by knowing the process from which to start monitoring execution behaviors.

Implementing File System Rules. To detect when $f1$, $f2$, and $f3$ occur by some process P, MiTCoN traces a set of file system functions located in the Windows kernel using a form of function hooking [5, 13]. These functions belong to the Zw family [14] and are located in the Windows SSDT table [11]. Windows provides several ways to create and modify files at the user level. When these commands get passed down to the kernel level, Windows merges them into a handful of Zw functions. MitCon traces two sequences of Zw function calls that successfully implement $f1$, and one sequence for $f2$ and $f3$. The first sequence to determine the occurrence of $f1$, infection via self replication, for some process P, MiTCoN traces the sequence of Zw functions with appropriate parameters listed in Table 1. The key to determining that $f1$ has occurred, is to establish that P is referencing itself and is the source of the write operation. According to Table 1, self-replication starts with P opening itself with read access in ZwCreateFile where sourcepath is the absolute path of P. The functions ZwCreateSection and ZwMapViewofSection use the file handle returned from ZwCreateFile to map the data that is going to be written from P into memory. Finally, ZwWriteFile reads the data stored in baseaddress, which is returned by ZwMapViewofSection, into the target file. At this point, an instance of $f1$ has been completed by P. Note in MapViewofSection, the in parameter processhandle is assured to be the value -1. This indicates the mapping will be of the caller process P. This sequence of calls is used when P makes an exact copy of itself in a newly created file.

```
ZwCreateFile(in:read_access, in:sourcepath, out: filehandle);
ZwCreateSection(in: filehandle, out:sectionhandle);
ZwMapViewofSection(in:sectionhandle,
   in:processhandle, out:baseaddress);
ZwWriteFile(in:baseaddress, out:target filepath)
```

Table 1. 1st Function Sequence Used in $f1$

The second sequence to determine the occurrence of $f1$ for some process P, MiTCoN traces the sequence of Zw functions with appropriate parameters listed in Table 2. The second sequence in determining $f1$ is simpler involving only two Zw functions. The source path in ZwReadFile refers to the absolute path of P and the memaddress is used as temporary storage of the data from P which is written to targetfilepath in ZwCreateFile. This sequence of calls is used primarily when P is self replicating into already existing files.

```
ZwReadFile(in:sourcepath, out:memaddress);
ZwWriteFile(in:memaddress, out:target filepath);
```

Table 2. 2nd Function Sequence Used in $f1$

To determine the occurrence of $f2$, infection via arbitrary file creation, for some process P, MiTCoN traces with appropriate parameters the ZwCreateFile function as listed in Table 1. The only parameter considered is sourcepath which is assured not to be the name and file system location of the caller process P. With this assurance, P, the caller process, is creating a completely new file with the name and file system location stored in sourcepath. To determine the occurrence of $f3$, infection via arbitrary file write modification, for some process P, MiTCoN traces with appropriate parameters the sequence of functions listed in Table 2, which is one of the sequences used for $f1$. The difference is in establishing an occurrence of $f3$, the sourcepath parameter in ZwReadFile is assured not to be the file system location of P. With this assurance $f3$ is determined since the data being written is from some other part of the system and not from P.

Implementing Process Rules. To detect when $p1$ and $p2$ occurs by some process P, MiTCoN performs function hooking on kernel level Zw functions and user level API functions. Several of the various ways in which a process can be injected and spawned at the user level filter down to a handful of Zw functions in the kernel. In Windows, a process can be spawned in two primary forms: a WIN32 process and a Windows service. MiTCoN traces one sequence of function calls for $p1$, and two sequences for $p2$. To determine the occurrence of $p1$, infection via dynamic code injection of a currently running process, for some process P, MiTCoN traces with appropriate parameters the sequence of function calls listed in Table 3. This sequence of functions facilitates the injection of data between process allocated memories. MiTCoN assures P is the caller of all three functions. Memory is first allocated with ZwAllocateVirtualMemory in the process identified by processhandle starting at baseaddress. these two parameters are again used to write (inject) data by P in the allocated memory of processhandle. Finally P creates a new instance of the just injected process with CreateThread which causes the new instance to execute the newly written nefarious code.

```
ZwAllocateVirtualMemory(in:processhandle, out:baseaddress);
ZwWriteVirtualMemory(in:processhandle, in:baseaddress);
ZwCreateThread(in:processhandle, out:threadhandle);
```

Table 3. Function Sequence Used in $p1$

The first sequence to determine the occurrence of $p2$, infection via process spawning, for some process P, MiTCoN traces with appropriate parameters the function call listed in Table 4. This single function suffices to create a new process from the static file image detailed in objectattributes. The function returns a handle, in processhandle, to the newly created process. MiTCoN checks if the newly created process is of a static file image that is already a node of P's MiT. If yes, then an edge is added.

```
ZwCreateProcess(in:objectattributes, out:processhandle)
```

Table 4. 1st Function Sequence Used in $p2$

The second sequence to determine the occurrence of $p2$ for some process P, MiTCoN traces with appropriate parameters the sequence of function calls listed in Table 5. CreateService will use the file located in BinaryPath as the service to be registered and returns in servicehandle a handle identifying the service. This handle is used in both OpenService and StartService, the end of which results in the service running on the system. Note the function CreateService has an additional parameter that, given the proper value, can start the service immediately. If this occurs, MiTCoN will not need to detect the invocation of OpenService and StartService.

```
CreateService(in:BinaryPath, out:servicehandle);
OpenService(inout: servicehandle);
StartService(in: servicehandle);
```

Table 5. 2nd Function Sequence Used in $p2$

MiTCoN assures P is the caller process for every function in a sequence by invoking GetCurrentProcess() at both the user and kernel levels. As rules are identified, the appropriate MiT is updated with new nodes and edges. The completed MiT presents all the files and processes identified as the source or destination of a given rule. The number of edges going in and out of any one node represents the number of rule instances.

MiTCoN Example. We present the MiT output of MiTCoN in Table 6 and its tree graph in Figure 2 for the malware Backdoor.Win32.Poison. Table 6 has three columns: Source identifies the caller process of the operation(s), Rule(s) gives the execution event rules invoked by Source, and Destination gives the target of the execution event. Each row is a complete invocation of a rule forming a node pair (source,destination) and Rule is the label for the outgoing edge. The first row's Source file name is the root of the MiT. In the first row, Poison performed rule $f1$ on svchest.exe, read as: Poison self-replicated into svchest.exe. The second to last row shows svchest.exe performed rules $f1$ and $p2$ on svchvst.exe, read as svchest.exe replicated into and spawned new process svchvst.exe. Poison has four child nodes: svchest.exe. 1.bat, 1.reg, and 1.vbs. Of these four, Poison self-replicated ($f1$) into svchest.exe and 1.vbs and created ($f2$) 1.bat and 1.reg.

Source	**Rule(s)**	**Destination**
Poison	$f1$	svchest.exe
Poison	$f2$	1.bat
Poison	$f2$	1.reg
Poison	$f1$	1.vbs
Poison	$p1$	wscript.exe
wscript.exe	$p1$	cmd.exe
cmd.exe	$p1$	regedit.exe
cmd.exe	$p1$	attrib.exe
cmd.exe	$p1$	reg.exe
cmd.exe	$p1$	reg.exe
cmd.exe	$p2$	svchest.exe
cmd.exe	$p1$	ping.exe
svchest.exe	$f1$,$p2$	svchvst.exe
svchvst.exe	$f1$	svchist.exe

Table 6. MiT of Backdoor.Win32.Poison

4 Evaluation & Results

Using MiTCoN, we analyzed and built MiTs for 5800 diverse known malware samples randomly selected from GFI Sandbox malware repository [4] uploaded between April and June 2011. According to Kaspersky anti-virus, the samples were classified as: Trojans, Worms, Viruses, Adware, Spyware, FakeAVs, Monitors, Risk Tools, PSW Tools, Hoaxes, web Tool Bars, Porn Dialers, Downloaders, Remote Admin Tools, IRC Clients, P2P worms, Email worms, Backdoors, Bots, Bankers, Clickers, Ransom, Packed, Game Thiefs, Exploits, Rootkits, and Droppers. Analysis was

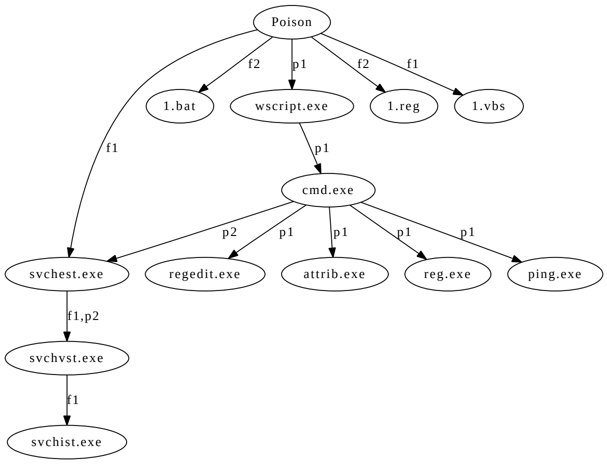


Figure 2. MiT graph of Backdoor.Win32.Poison

conducted in VMWare Workstation with a fresh install of Microsoft Windows Vista logged into the default administrator user account. MiTCoN and each sample were executed for three minutes, the MiT enhanced with timestamps was saved to a database and the snapshot reverted to a clean state. MiTCoN proved to be highly efficient by not causing system instability or crashes during analysis, CPU usage per sample peaked at 3% and averaged less than 1%. Total RAM memory usage never surpassed 14MB with MiTCoN executing. MiTCoN never took more than 7 seconds to build and record a MiT and averaged these operations at 3.1 seconds. There was a high frequency of rule occurrence in all analyzed malware samples with overall totals as follows: $f1$:662, $f2$:14396, $f38629$, $p1$:647, $p2$:3490 with the top three malware classes: Trojan: 293, Worm:63, Backdoor:56. Every analyzed sampled had multiple occurrences strong bond rules implying strongly bonded MiTs may suffice to understand the malware's essential infection strategy with weak bonds enhancing the MiT. In the instances of $p1$, we infered the malware was delegating nefarious deeds off to seemingly benign processes in order to achieve their goals while not being identified. The majority of processes were spawned as WIN32 with a several being pre-existing nodes in the MiT itself ($p2$). Many files where spawned as a Windows service and not a regular WIN32 process. Detecting services proved critical to building comprehensive trees since the node's subtree was substantial. Many nodes had multiple incoming and/or outgoing edges which illustrates a strong bond between file and process manipulation by malware. The most common combination was $f1, p2$ where malware would self replicate then spawn the process to have multiple instances running on the machine. We conjecture this avoids complete malware eradication or overwhelms the system facilitating compromise. Analyzing timestamps of rule occurrence, we discovered all instances of every rule occurred within 200 milliseconds from initial malware execution with an overall average 12 milliseconds, the strong bond rules averaged 11 milliseconds for $f1$ and 14 milliseconds for $p1$. We randomly selected 120 malware samples and executed them on the analysis platforms Anubis and GFI SandBox. The resulting infection tree structures from both were compared to the MiT's created by MiTCoN. In 114 samples both platforms included nodes (either a process or file) that was excluded in our MiT. Further analysis revealed these nodes represented files and processes belonging to standard Windows process operations and were not part of the malware infection and are therefore false positives. A typical scenario was the inclusion of services.exe, which is used in Windows each time a process requests creation or start of a Windows service. Our approach to MiT building based on fundamental malware infection seems to create more relevant MiTs with the ability to exclude files and processes belonging to standard Windows operations. The high frequency, early occurrence and lack of false positives makes our rules for building MiTs highly effective in analyzing malware.

We test the possible usability of MiTCoN in system disinfection with our previously randomly chosen 120 samples and use the resulting MiT to attempt disinfection of the system. Testing was done in VMWare Workstation running a snapshot of a fresh install of Microsoft Windows Vista logged into the default administrator user account. Kaspersky anti-virus [6] scanned the infected system to assess how successful our disinfection attempt was. Before testing, we invoked a complete system scan by Kaspersky which resulted with no infections found. This was done to assure an initial malware free testing environment and any discovered infections occurring after this initial scan was attributed to the malware executed by us in the test system. Our evaluation was performed in two rounds. Round one of testing was as follows: 1.Initially, the clean state snapshot is loaded 2.A malware sample is copied to the Windows desktop 3.MiTCoN is executed using the sample's path as input 4.The sample is executed for 3 minutes 5.The resulting MiT is saved for later use 6.The infected snapshot is scanned with Kaspersky & the results saved 7.Return to step 1 with next sample. The first round of testing was performed to create MiTs for each of our test samples and to assure Kaspersky can detect the malware infection. In every case, Kaspersky detected the malware. Knowing Kaspersky can detect our malware sample's infection was a pre-requisite to the second round of testing where Kaspersky is used to assess the effectiveness of our disinfection attempt. Round two of testing was as follows: 1.Initially, the clean state snapshot is loaded 2.A malware sample is copied to the Windows desktop 3.The sample is executed for 3 minutes 4. The system is manually disinfected using the samples's MiT from round one 5.The snapshot is scanned with Kaspersky & the results saved 6.Return to step 1 with next malware sample. The main purpose of round two was to assess how effective our disinfection attempt was using a MiT. In each case the files and processes listed in the MiT that were found in the infected system were removed. Kaspersky did not detect any malicious objects in the second round of testing implying our MiTs were effective in eradicating infection from the system. In every case, the details in the MiT sufficed to erradicatd the files and processes from the system.

Limitations. Our testing was conducted in a virtual machine which forcibly excluded using vm-aware malware. MiTCoN is limited by the number of implementations in which a rule can be traced in a specific OS. We are continuously adding new OS specific implementations of a rule into MitCoN as well as discovering new rules reflecting malware infection strategies. Secure testing on actual machines is also being crafted for future use.

5 Conclusion & Future Work

We have presented an abstract approach to building comprehensive MiTs based on rules describing execution events essential to malware infection strategies of files and processes. We developed a MiT creation tool, named MiTCoN, for the Windows Vista platform and tested with 5800 known malware samples. Our analysis revealed MiTCoN was very efficient in MiT building and our rules were highly effective in identifying the relevant malware infection files and processes with high occurrence rate in all samples completing in under 200 milliseconds. In contrast to similar systems, our MiTs avoid false positives by correctly excluding non-malware related processes and files. The MiTs produced by our tool during analysis produced 100% successful manual system disinfection verified with a post-disinfection malware scan. Our results suggests our abstract approach using a malware infection strategy basis for rule creation produces MiTs which are highly effective, improve analysis and disinfection, and produce minimal false positives. Our future work includes adding new execution event rules, secure testing in a real machine and the ability to create MiTs for malware samples that are not WIN32 executables.

Acknowledgements. This work is partially supported by grants from AFOSR, ONR, AFOSR MURI, and the State of Texas Emerging Technology Fund.

References

[1] L. Adleman. An abstract theory of computer viruses. In *CRYPTO '88: Advances in Cryptology*, pages 354 374. Springer, 1988.

[2] F. Cohen. *A Short Course on Computer Viruses*. Wiley Professional Computing, 1994. ISBN 0-471-00769-2.

[3] E. Filiol. *Computer Viruses: from Theory to Applications*. IRIS International series, Springer Verlag, 2005. ISBN 2-287-23939-1.

[4] http://www.gfi.com/malware-analysis-tool/.

[5] G. Hoglund and J. Butler. *Rootkits: subverting the Windows Kernel*. Addison Wesley Professional, 2005.

[6] Kaspersky anti-virus. http://www.kaspersky.com.

[7] N. Kawaguchi, H. Shigeno, and K.-i. Okada. Detection of silent worms using anomaly connection tree. In *Proceedings of the 21st International Conference on Advanced Networking and Applications*, AINA '07, pages 412 419, Washington, DC, USA, 2007. IEEE Computer Society.

[8] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 351 366, Berkeley, CA, USA, 2009. USENIX Association.

[9] J. A. Morales, P. J. Clarke, Y. Deng, and B. G. Kibria. Identification of file infecting viruses through detection of self-reference replication. *Journal in Computer Virology Special EICAR conference invited paper issue*, 2008.

[10] V. Skormin, A. Volynkin, D. Summerville, and J. Moronski. Prevention of information attacks by run-time detection of self-replication in computer codes. *Journal of Computer Security*, 15(2):273 302, 2007.

[11] System service dispatch table. `http://en.wikipedia.org/wiki/System_Service_Dispatch_Table`.

[12] P. Szor. *The Art of Computer Virus Research and Defense*. Symantec Press and Addison-Wesley, 2005. ISBN 9-780321-304544.

[13] R. Vieler. *Professional Rootkits*. Wrox Press, 2007.

[14] Zwxx routines in windows kernel. http://msdn.microsoft.com/en-us/library/ms804352.aspx.

6 Appendix B: Reports of the Subcontracts of the Supplementary Equipment Grant

This Appendix includes:

- Final Report of Binghamton University Equipment Subcontract Grant
- Final Report of Louisiana Tech University Equipment Subcontract Grant
- Final Report of Tennessee State University Equipment Subcontract Grant
- Final Report of UIUC Equipment Subcontract Grant
- Final Report of University of South Carolina Equipment Subcontract Grant

Final Report of Binghamton University Equipment Subcontract Grant

PI: Kyoung-Don Kang

In this project, we have investigated research issues related to networking monitoring and stream data analysis that is a basis for the security and maintenance of the cloud computing infrastructure. **Data stream management systems** (DSMS) are developed to efficiently process data streams in important applications, e.g., network monitoring, financial market analysis, and transportation management. As data streams may be unbounded and arrive at high rates, most DSMS support continuous reevaluation of queries over sliding (or jumping) windows of stream data tuples, e.g., network monitoring data or stock price data streams. Using a declarative language provided by a DSMS, a user can easily write a DSMS application, which is interpreted as a graph of the interconnected continuous query (CQ) operators necessary to process data streams for the application.

Ideally, DSMS are desired to process data streams with a minimal delay to monitor critical real world phenomena, e.g., market status changes or distributed denial of service (DDoS) attacks. Ironically, supporting this property is most challenging when it is most needed. For example, a large amount of data streams may arrive in a short time interval when the status of a financial market or communication network is widely fluctuating. Thus, the DSMS can be overloaded, resulting in late detection of DDoS attacks or losses of business opportunities. To deal with potentially bursty arrivals of stream data, a CQ operator, e.g., a windowed select, project, join, or aggregation operator, usually buffer its input data streams in memory. Under overload, however, the input stream queues for buffering may grow rapidly. As a result, the system may suffer from excessive data losses or CQ processing delays. It may even become unreliable due to potential thrashing caused by severe memory contention.

To address the problem, we have designed a novel load shedding scheme to systematically control the backlog accumulated by each stream even in the presence of dynamic workloads. For load shedding, a DSMS should determine 1) when to shed load, 2) where to shed load, 3) which stream data tuples to drop, and 4) how much load to shed [7]. Concerning 1) and 2), we take an immediate, localized approach to load shedding. Our load shedding scheme runs immediately when the amount of backlog of any individual data stream changes by more than a specified threshold. It probabilistically drops tuples, if necessary, to control per-stream backlog to be below the desired bound without requiring any global coordination, which is computationally expensive and time consuming in distributed DSMS. By doing this, our approach avoids a potential avalanche, in which the backlog accumulated in a stream queue may largely delay the succeeding CQ operators in a cascading manner. Further, the impact of tuple dropping is evenly spread across the query operators cooperating for a specific application. If load shedding is applied only to the data stream sources, too many queries can be affected, since one stream may fan out to multiple streams [7]. On the other hand, only performing load shedding at the sinks of the query graph, which produce the final CQ processing results, is subject to excessive resource waste caused by a large amount of intermediate data streams that will be eventually dropped.

Regarding 3) and 4), we drop tuples randomly according to the drop probability, because random tuple dropping incurs a minimal decrease of the accuracy of query processing results without making any statistical assumptions about data distributions. To determine how much load to shed, we apply fuzzy control techniques [9] to derive the drop probability considering the current DSMS status. Fuzzy control is based on fuzzy logic developed to deal with uncertainties prevalent in real world applications. Due to its robustness, it is widely applied to support performance and safety features in many important systems including automobiles and consumer electronics. It can naturally express fuzzy (uncertain) concepts that are not completely true or false but partially true. Thus, it is very effective to handle uncertainties in highly-dynamic, nonlinear systems. In this project, we apply it to efficiently handle uncertainties in DSMS including bursty data arrivals and aperiodic, data-dependent CQ executions. We have implemented our approach and compared its performance to that of Borealis—one of the most advanced open-source DSMS using **CAIDA data** comprised of real-world network traffic traces tailored for research. Our approach showed **significant performance improvement** in terms of the delay and data stream backlog.

In addition to the DSMS research, we have designed and developed a **GPU-accelerated MapReduce** framework for high performance big data analysis. For more details of our work on data stream management systems and a GPU-accelerated MapReduce, please refer to the attached papers [1,2] that fully discuss the results of the work. These research results are accepted to the *IEEE International Conference on Service Oriented Computing & Applications (SOCA '12)* and submitted to the *Journal of Parallel and Distributed Computing.*

References

[1] C. Basaran, K. D. Kang, Y. Zhou, and M. H. Suzer, "Adaptive Load Shedding via Fuzzy Control in Data Stream Management Systems," *IEEE International Conference on Service Oriented Computing & Applications (SOCA '12),* Dec. 17 - 19, 2012, Taipei, Taiwan, To Appear.

[2] C. Basaran and K. D. Kang, Grex: An Efficient MapReduce Framework for Graphics Processing Units, Journal of Parallel and Distributed Computing (Under Review)

Adaptive Load Shedding via Fuzzy Control in Data Stream Management Systems

Can Basaran*, Kyoung-Don Kang†, Yan Zhou‡ and Mehmet H. Suzer§
*Dept. of Inf. and Comm. Eng., Daegu Gyeongbuk Institute of Science & Technology (DGIST), Korea
Email: cbasaran@dgist.ac.kr
†Dept. of Comp. Sci., Binghamton University, Email: kang@binghamton.edu
‡Cisco Systems, Email: yazhou2@cisco.com
§Dept. of Comp. Sci., Harran University, Email:msuzer@harran.edu.tr

Abstract—**Data stream management systems (DSMS) aim to process massive data streams in a timely fashion to support important applications, e.g., financial market analysis. However, DSMS can be overloaded due to large bursts in data stream arrivals and data-dependent query executions. To avoid overloads, we design a new load shedding scheme by applying distributed fuzzy logic control, which is very effective to deal with uncertainties in highly dynamic systems such as DSMS, based on the per-stream backlog and selectivity of each query operator. We have implemented our approach by extending an open source distributed DSMS. The performance evaluation using high-rate Internet traces shows that our approach closely supports a specified backlog bound for each data stream queue, while improving the query processing delay, with little overhead.**

I. Introduction

Data stream management systems (DSMS), including [1], [2], [3], [4], [5], [6], are developed to efficiently process data streams in important applications, e.g., network monitoring, financial market analysis, and transportation management. As data streams may be unbounded and arrive at high rates, most DSMS support continuous reevaluation of queries over sliding (or jumping) windows of stream data tuples, e.g., stock price or network monitoring data streams. Using a declarative language (e.g., a windowed SQL-like language [5] or graphical query processing boxes [4]) provided by a DSMS, a user can easily write a DSMS application, which is interpreted as a graph of the interconnected continuous query (CQ) operators necessary to process data streams for the application.

Ideally, DSMS are desired to process data streams with a minimal delay to monitor critical real world phenomena, e.g., market status changes or denial of service (DoS) attacks. Ironically, supporting this property is most challenging when it is most needed. For example, a large amount of data streams may arrive in a short time interval when the status of a financial market or communication network is widely fluctuating. Thus, the DSMS can be overloaded, resulting in late detection of DoS attacks or losses of business opportunities. To deal with potentially bursty arrivals of stream data, a CQ operator, e.g., a windowed select, project, join, or aggregation operator, usually buffer its input data streams in memory. Under overload, however, the input stream queues for buffering may grow rapidly. As a result, the system may suffer from excessive data losses or CQ processing delays. It may even become unreliable due to potential thrashing caused by severe memory contention.

To address the problem, we design a novel load shedding scheme to systematically control the backlog accumulated by each stream even in the presence of dynamic workloads. For load shedding, a DSMS should determine 1) when to shed load, 2) where to shed load, 3) which stream data tuples to drop, and 4) how much load to shed [7]. Concerning 1) and 2), we take an **immediate, localized** approach to load shedding. Our load shedding scheme runs immediately when the amount of backlog of any individual data stream changes by more than a specified threshold. It probabilistically drops tuples, if necessary, to control per-stream backlog to be below the desired bound without requiring any global coordination, which is computationally expensive and time consuming in distributed DSMS. By doing this, our approach avoids a potential avalanche, in which the backlog accumulated in a stream queue may largely delay the succeeding CQ operators in a cascading manner. Further, the impact of tuple dropping is evenly spread across the query operators cooperating for a specific application. If load shedding is applied only to the data stream sources, too many queries can be affected, since one stream may fan out to multiple streams [7]. On the other hand, only performing load shedding at the sinks of the query graph, which produce the final CQ processing results, is subject to excessive resource waste caused by producing a large amount of intermediate data streams that will be eventually dropped.

Regarding 3) and 4), we drop tuples **randomly** according to the drop probability, because random tuple dropping incurs a minimal decrease of the accuracy of query processing results without making any statistical assumptions about data distributions [5], [7].[1] To determine how much load to shed, we apply **fuzzy control** techniques [9] to derive the drop probability considering the current DSMS status. Fuzzy control is based on fuzzy logic developed to deal with uncertainties prevalent in real world applications. Due to its robustness, it is widely applied to support performance and safety features in many important systems including automobiles and consumer electronics. Since it can naturally express fuzzy (uncertain)

[1] Our approach is not limited to random dropping. If the utilities of the data streams are known *a priori* similar to [8], our approach can be easily adapted to drop the tuple with the smallest utility first.

concepts that are not completely true or false but partially true, it is very effective to handle uncertainties in highly-dynamic, nonlinear systems. In this paper, we apply it to efficiently handle uncertainties in DSMS including bursty data arrivals and aperiodic, data-dependent CQ executions.

Fuzzy control is performed using if-precedent-then-consequent rules derived based on the logical understanding of the fundamental characteristics of the controlled system, e.g., a DSMS. We design a set of novel fuzzy rules to dynamically adjust the **per-stream** drop probability, if necessary, to control the length of each stream queue to be below the specified set-point based on the queue length error, i.e., the difference between the queue length set-point and current length of any individual data stream queue in a DSMS, even in the presence of dynamic workloads. Moreover, we design another set of new fuzzy rules to consider not only the queue length error but also the dynamics of query operators to drop **just enough tuples**, if necessary, to avoid overloads. Specifically, we consider the selectivity of each individual query operator, i.e., the ratio of the number of the output stream tuples produced by an operator to the number of the input stream tuples entering the operator, to further adjust the shedding probability. We seamlessly integrate the per-stream queue length and selectivity controllers into a single load shedding framework to avoid both overloads and excessive data losses in DSMS. For example, even when a data stream queue is longer than the desired bound, our approach does not increase the shedding probability for the specific stream, if a large fraction of the stream is filtered out by a preceding query operator before the stream reaches the queue being controlled (or vice versa).[2]

In addition, our fuzzy logic scheme for load shedding is executed in an **event-driven** manner; that is, it is executed when the length of any queue, i.e., per-stream backlog, changes by more than the specified event threshold. Being event-driven, our load shedding scheme automatically executes more often when the per-stream backlog increases or vice versa unlike the other control theoretic techniques using a fixed control period [10], [11], [12]. Further, our approach requires neither centralized control nor global information about the system status. For each stream queue, the shedding probability is adjusted considering the per-stream queue size error and selectivity of the preceding CQ operator. In summary, our approach supports immediate, localized per-stream backlog control in an event-driven manner to control the size of each data stream buffer to be below a specified set-point even in the presence of uncertainties. Supporting these features is not directly considered by previous work on DSMS including [5], [4], [6], [13], [14], [15], [8], [16].

For performance evaluation, we have implemented our approach by extending one of the most advanced open source DSMS, Borealis [4], in a Linux cluster. For performance comparisons between our approach and Borealis, we use high-speed Internet traces [17] different from most existing research on DSMS (including [5], [4], [6], [18], [19], [13], [14], [15], [8], [16]), which uses synthetic workloads or outdated traces such as [20], [21]. The performance evaluation results show that our approach closely supports the desired per-stream queue length set-point. The per-stream queue length of our approach is up to two orders of magnitude shorter than the one supported by Borealis [16]. In addition, our approach considerably decreases the delay for query processing.

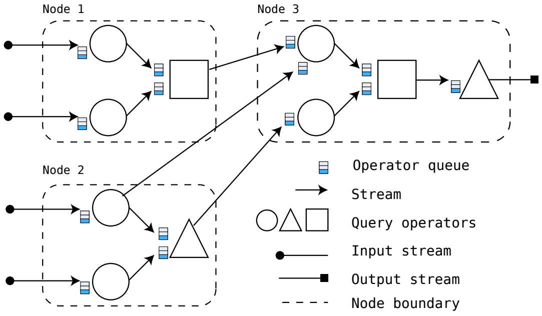


Fig. 1. A Continuous Query Tree

Notably, our approach is *lightweight*. It consumes less than 1% CPU utilization and 1KB of memory to store the if-then rules for systematic load shedding. It has little overhead, since it is localized and requires no global feedback or adaptation across the entire query graph (also called a query network in the DSMS literature). Further, it only requires simple computations and look-ups of the fuzzy rules stored in two tables to compute the tuple drop probability for load shedding.

The rest of the paper is organized as follows. An overview of the DSMS architecture and load shedding in DSMS is given in Section II. A description of our load shedding scheme and fuzzy rule design is given in Section III. Performance of our approach is evaluated in Section IV. Section V discusses related research. Finally, Section VI concludes the paper and discusses future work.

II. System Architecture

In a distributed DSMS application, the stream sources provide input data streams to the rest of the nodes and the final output is produced at one or more sink nodes as shown in Figure 1. CQ operators, e.g., windowed select, project, or join, process data streams and produce intermediate data streams. A query is a tree of CQ operators. One or more queries can be linked together to form a query graph/network as shown in Figure 1. In DSMS, a queue is usually assigned to buffer data arriving from an individual data stream [4], [22], [5]. The stream queues in one node of a distributed DSMS usually share a single memory pool in the node without capping the length of an individual queue. Thus, a sudden arrival rate increase in a single stream can cause the other operators to starve due to the insufficient memory. As a result, the overall response time may increase due to the data and control dependencies in the query network.

[2] Note that network congestion control is not directly applicable to load shedding in DSMS [7], since it simply drops packets as the buffer in a network router builds up without simultaneously considering the aforementioned four issues and CQ selectivity important in DSMS.

To prevent a subset of streams from consuming excessive amounts of memory, we directly control the backlog accumulated in each stream queue to be below a specified threshold, Q_S, even in the presence of bursty stream data arrivals and aperiodic, data-dependent executions of query operators.[3] Since the arrival rate and selectivity may vary from stream to stream and operator to operator, our approach to load shedding performs on a per-stream, per-CQ operator basis.

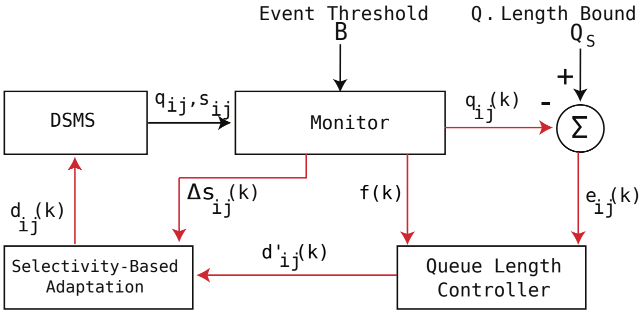


Fig. 2. Adaptive Load Shedding

Figure 2 shows the high level structure of our adaptive load shedding scheme. Let I_{ij} represent stream j flowing into CQ operator i in a distributed DSMS. In Figure 2, q_{ij} represents the instantaneous length of the queue used to buffer I_{ij}'s tuples and $q_{ij}(k)$ represent the size of the queue at the k^{th} control event that occurs when:

$$|q_{ij}(k) - q_{ij}(k-1)| \geq B \tag{1}$$

where $k \geq 1$ and B is the control event threshold used to detect a noteworthy increase or decrease of q_{ij}. (We assume that $q_{ij}(0) = 0 \ \forall i,j$ when the system starts initially.) We take an event-driven approach, since it is infeasible to find one common control period effective to monitor and adapt all streams, which may have significantly different average and transient arrival rates.

To control the length of each stream queue to be below the specified set-point Q_s, we employ two interconnected controllers: a queue length control module (QCM) and a selectivity-based control module (SCM) in Figure 2. Via fuzzy control, QCM computes the drop probability $d'_{ij}(k)$. SCM computes $d_{ij}(k)$ actually used to probabilistically drop tuples of I_{ij}, if necessary, to support Q_s. To compute $d_{ij}(k)$, SCM adjusts $d'_{ij}(k)$ considering the selectivity of the preceding CQ operator, which pushes stream data tuples into I_{ij}'s queue.

Whenever a control event occurs for an arbitrary stream in a DSMS, we apply our load shedding method to the specific stream. Therefore, in the rest of this paper, we consider only a single stream I_{ij} to present our per-stream load shedding scheme. We omit the operator and stream indexes i and j for brevity without loss of generality.

At the k^{th} (control) event, i.e., when $|q(k) - q(k-1)| \geq B$, **QCM** computes the **error**:

$$e(k) = (Q_s - q(k))/Q_s. \tag{2}$$

At the k^{th} event, QCM in Figure 2 measures the time between two consecutive events:

$$\Delta t(k) = t(k) - t(k-1) \tag{3}$$

where $t(k)$ indicates the time at which the k^{th} event occurs. QCM also computes the queue length change between the consecutive events:

$$\Delta q(k) = q(k) - q(k-1). \tag{4}$$

Using Eq 3 and Eq 4, QCM calculates the **trend** of the queue length change representing the speed and direction of the queue length change:

$$f(k) = \begin{cases} \Delta t(k) & \text{if } \Delta q(k) > 0; \\ -\Delta t(k) & \text{if } \Delta q(k) < 0 \end{cases} \tag{5}$$

Note that $\Delta q(k) \neq 0$ when $f(k)$ is computed, since a control event occurs only when the queue length changes by more than B as defined in Eq 1. By considering the trend in addition to the error, we intend to improve the effectiveness of per-stream backlog management in DSMS. For example, even when the queue length is longer than Q_s, it is unnecessary to increase the tuple drop probability by a large amount, if the queue length is decreasing fast or vice versa. Logical observations such as these ones can be easily formulated via fuzzy rules. Based on $e(k)$ and $f(k)$, QCM computes the drop probability $d'(k)$ using the fuzzy rules (described in Section III).

As shown in Figure 2, the output of QCM, $d'(k)$, is an input to **SCM** that adapts $d'(k)$, if necessary, to further enhance the effectiveness of load shedding by considering the selectivity. At the k^{th} event, SCM in Figure 2 computes the **selectivity change** of the preceding operator as follows:

$$\Delta s(k) = s(k) - s(k-1) \tag{6}$$

In this equation, $s(k)$ is the selectivity of the preceding operator computed at the k^{th} event. Thus, $0 \leq s(k) \leq 1$. Based on $\Delta s(k)$, SCM modifies $d'(k)$, if necessary, to expedite the convergence of the queue length to Q_s based on the selectivity information. For example, $\Delta s(k) < 0$, if the selectivity of the operator pushing a data stream into the queue is decreasing. In this case, SCM adjusts the drop probability to shed fewer incoming stream tuples by making $d(k) < d'(k)$ (or vice versa). In this way, it intends to avoid unnecessary data losses by dropping just enough tuples needed to closely support Q_s.

III. Application of Fuzzy Control to Manage Per-Stream Backlog

Figure 3 shows the structure of our closed-loop load shedding scheme consisting of QCM and SCM. In fuzzy control, crisp variables expressed as real numbers, e.g., $e(k)$ and $f(k)$ in Figure 3, are mapped to linguistic variables, e.g., the error and trend, to naturally express human thoughts via the process called fuzzification. Using the linguistic variables,

[3] For the clarity of presentation, we use a system-wide queue length bound Q_S that applies to every data stream queue in a DSMS. However, our approach is not limited to a single queue length bound. Different streams may have different queue length bounds.

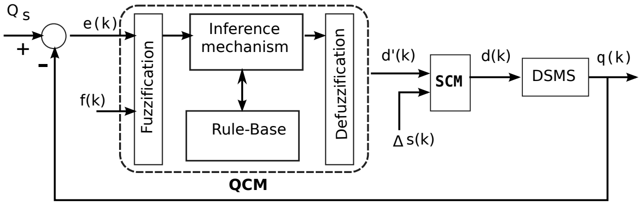


Fig. 3. Load Shedding via Fuzzy Logic Control

the inference scheme looks up the relevant if-precedent-then-consequent rules. Finally, the defuzzification interface, such as the one in Figure 3, derives a crisp control signal from the linguistic rules.[4]

In the rest of this section, we describe the key observations that motivated the design of our fuzzy if-then rules for effective per-stream load shedding. Also, we describe the detailed procedure for fuzzy control, while giving illustrative examples.

A. Design of Fuzzy Rules for Adaptive Load Shedding

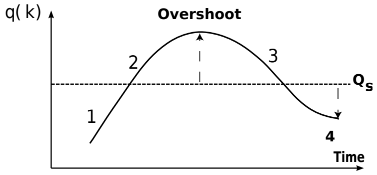
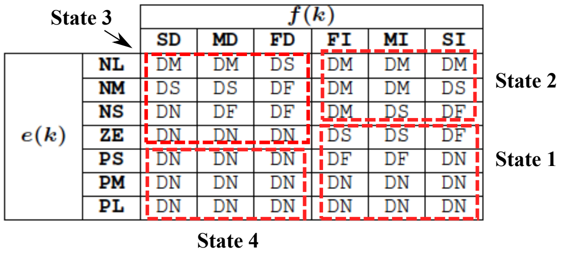


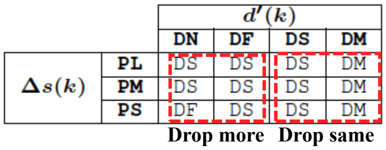
Fig. 4. Queue Length Variations



		$f(k)$					
		SD	MD	FD	FI	MI	SI
$e(k)$	NL	DM	DM	DS	DM	DM	DM
	NM	DS	DS	DF	DM	DM	DS
	NS	DN	DF	DF	DM	DS	DF
	ZE	DN	DN	DN	DS	DS	DF
	PS	DN	DN	DN	DF	DF	DN
	PM	DN	DN	DN	DN	DN	DN
	PL	DN	DN	DN	DN	DN	DN

State 3
State 2
State 1
State 4

Fig. 5. Rule-base of QCM used to derive $d'(k)$ from $e(k)$ and $f(k)$



		$d'(k)$			
		DN	DF	DS	DM
$\Delta s(k)$	PL	DS	DS	DS	DM
	PM	DS	DS	DS	DM
	PS	DF	DS	DS	DM

Drop more
Drop same

Fig. 6. Rule-base of SCM used to derive $d(k)$ from $\Delta s(k)$ and $d'(k)$

For effective fuzzy control, it is essential to design an effective *rule-base*. They are designed based on the logical understanding of the characteristics of the controlled system such as a DSMS. Figures 5 and 6 are the rule-bases used by QCM and SCM, respectively. The rules are designed to support Q_s even when the system status and workloads change dynamically. For example, the rule $\{NS, FI\} \rightarrow DM$ in Figure 5 requires the load shedder to drop most (DM) tuples of the stream (I_{ij}), if the current error, i.e., the linguistic version of $e(k)$, is negative small (NS), i.e., $e(k) > Q_s$ by a small amount. However, the trend, i.e., the linguistic version of $f(k)$, is fast increasing (FI), i.e., $f(k) \gg 0$.

We design the rules in Figure 5 based on our observations depicted in Figure 4. In the figure, there are four major states that characterize dynamic queue length changes. In normal conditions, it is desired that the system is in states 1 or 4 in Figure 4. Queue length overshoots are observed in states 2 and 3; that is, the queue length is longer than Q_s. Also, state 2 is worse than state 3, since the queue length is diverging from Q_s in state 2, whereas it is converging to Q_s in state 3. The objective of QCM is to avoid overshoots by adapting the drop probability based on the error and trend, which are the linguistic (i.e., fuzzified) versions of Eq 2 and Eq 5. More specifically, we design the rule-base in Figure 5 for **QCM** considering the four states in Figure 4 as follows.

State 1. $e(k) \geq 0$ and $f(k) > 0$: In this state, the queue is not longer than Q_S, but it is increasing. Thus, the linguistic error is zero or positive, while the trend is increasing.

- Case 1: The error is zero. In this case, the queue length is equal to Q_s. If the trend indicates a fast increase (FI) or medium (MI) of the error, we drop some (DS) tuples as shown in in Figure 5. On the other hand, we drop few (DF) tuples, if the trend is a small increase (SI).
- Case 2: The error is positive. In this case, the queue is shorter than Q_s. As this is a desirable state, we either drop few (DF) tuples or drop none (DN). If the error is positive small (PS), i.e., the queue is shorter than Q_s by a small amount, and the trend is FI or MI, we drop few (DF) tuples. Otherwise, the rules recommend the DSMS to drop none (DN).

State 2. $e(k) < 0$ and $f(k) > 0$: In this zone, the queue is longer than Q_S and further increasing. To cancel the overshoot and divergence from Q_s, QCM recommends the DSMS to drop the largest fraction of tuples in this state. It derives the tuple drop probability considering the magnitudes of the error and trend. For example, when the error is negative medium (NM), the queue is longer than Q_s by a medium amount. In this case, the rules $\{NM, SI\} \rightarrow DS$ and $\{NM, MI\} \rightarrow DM$ in Figure 5 require the DSMS to drop some (DS) and drop most (DM) tuples for the slow increase and medium increase of the queue length, respectively.

State 3. $e(k) \leq 0$ and $f(k) < 0$: In this zone, the queue length is longer than or equal to Q_s, but it is decreasing. Control rules should be carefully designed by comparing the magnitudes of the error and trend, since it is possible for them to cancel each other. For example, if the error is negative small (NS) and the queue length shows a slowly decreasing (SD) trend, no tuple will be dropped as shown in Figure 5. In this region, a higher drop probability is derived for a larger absolute

[4] The fuzzification/defuzzification interface, inference scheme, and rule-base of SCM are not shown in Figure 3 to avoid a graphical repetition.

value of the error and a slower decrease of the queue length or vice versa.

State 4. $e(k) > 0$ and $f(k) < 0$: In this state, the actual queue length is shorter than the set-point and it is further decreasing. Since this is a desirable state, the QCM requires the DSMS to drop none of the tuples.

SCM aims to further decrease the possibility of a queue length overshoot. To achieve the objective, SCM uses the rule-table in Figure 6 to derive $d(k)$ based on $\Delta s(k)$ and $d'(k)$ computed by QCM. More specifically, SCM amplifies $d'(k)$ only when the selectivity is increasing, i.e., $\Delta s(k) = s(k) - s(k-1) > 0$. In this case, SCM amplifies $d'(k)$ based on the magnitude of $\Delta s(k)$ and $d'(k)$. Thus, the rule-base in Figure 6 only considers positive selectivity changes.

For example, suppose that the selectivity change is positive large (PL), but QCM produces a drop none (DN) signal. In this case, the rule $\{PL, DN\} \rightarrow DS$ in Figure 6 modifies the DN signal produced by the QCM to the drop some (DS) signal, because the preceding operator's selectivity is increasing largely and the length of the stream queue succeeding the operator is likely to increase as a result.

Notably, the first two columns in Figure 6 require the system to further increase the drop probability considering the selectivity increase. On the other hand, the last two columns require no further increase of the drop probability, since the probability computed by QCM is already high enough in these cases. In this way, SCM aims to avoid overshoots and excessive data losses simultaneously.

B. Fuzzy Control Procedure

In this section, the fuzzification, inference, and defuzzification process for QCM is described. The procedure used by SCM is similar and omitted due to space limitations.

At the k^{th} control event, the fuzzification interface of QCM in Figure 3 maps $e(k)$ and $f(k)$ to the linguistic variables of the error and trend with the corresponding linguistic values. Using the derived linguistic values, the inference mechanism in in Figure 3 looks up the rule-base to find relevant if-then rules. From the relevant rules, the defuzzification interface of QCM derives a crisp control signal $d'(k)$, via defuzzification, to support the desired per-stream backlog bound Q_s.

For fuzzy control, each variable, such as the variables in Figures 7(a) $-$ 7(c), should be associated with the *universe of discourse*, i.e., the domain, [9]. In this paper, the universe of discourse for $e(k)$ and $\Delta s(k)$ is [-0.5, 0.5], while the universe of discourse for $f(k)$ is [-3.5, 3.5].[5] For the control output, i.e., $d'(k)$ and $d(k)$, we use the universe of discourse of [0, 0.95] to bound the range of the drop probability.

In fuzzy set theory underlying fuzzy control theory, to deal with **uncertainties**, a crisp value is mapped to one or more fuzzy sets; that is, *partial membership* is allowed unlike set theory that only supports binary membership [9]. The degree of membership, called *certainty*, is computed using membership functions (MFs). In this paper, $e(k)$ may belong to one or two fuzzy MFs in Figure 7(a), because we design adjacent MFs to overlap. For all the MFs in Figure 7(a), except for the leftmost and rightmost ones, we use symmetric triangles of an equal base and 50% overlap with the adjacent fuzzy MFs. In Figure 7(a), the horizontal axis represents $e(k)$, while the vertical axis indicates the certainty of the membership.

Fuzzification is the first step in fuzzy control. It maps the input crisp variables to the corresponding linguistic variables. In this paper, $e(k)$ measured at the k^{th} control event are fuzzified to the linguistic error with one or two linguistic values in Figure 7(a): Negative Large, Negative Medium, Negative Small, ZEro, Positive Small, Positive Medium, and Positive Large. If $e(k) < 0$, it is mapped to one or more Negative MFs in Figure 7(a) or vice versa.

Via fuzzification, we also map $f(k)$ to the linguistic trend with one or two of the linguistic values in Figure 7(b): Slow Decrease, Medium Decrease, Fast Decrease, Slow Increase, Medium Increase, and Fast Increase. If $f(k) < 0$, it is mapped to one or two Decrease MFs in Figure 7(b) or vice versa (Eq 5). In the following example, QCM fuzzifies $e(k)$ and $f(k)$.

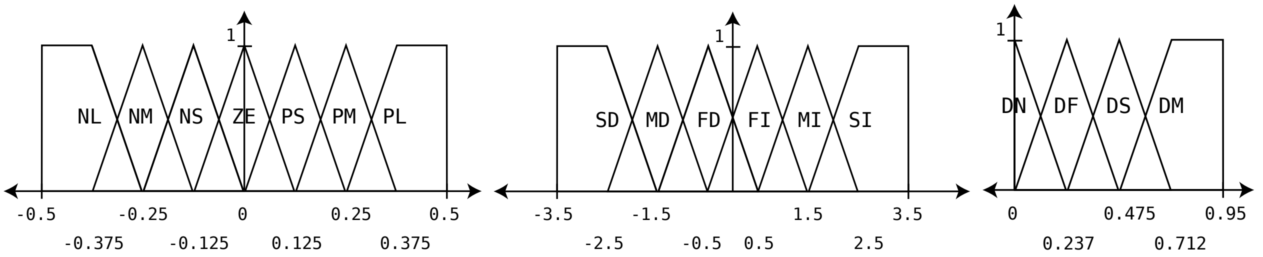
Example 1. Fuzzification. Let $e(k) = -0.15625$ and $f(k) = 1.5$. In this case, the error, i.e., the linguistic version of $e(k)$, belongs to NM and NS in Figure 7(a) with certainty $\mu_{NM}(-0.15625) = 0.25$ and $\mu_{NS}(-0.15625) = 0.75$. At the same time, the trend, i.e., the linguistic version of $f(k)$, belongs to MI in Figure 7(b) with certainty 1.

Using the fuzzified linguistic error and trend, the inference mechanism of QCM looks up the rule-base in Figure 5 to find the related if-then rules. The linguistic variables are used to infer the if-then rules. Due to partial membership allowed in fuzzy control, *more than one if-then rules can be triggered* as a result of the rule-base look-up. In this case, the certainty of an inferred rule is set to the *minimum* of the certainty values of the precedents, because the consequent cannot be more certain than the precedents [9].

Example 2. Rule-Base Lookup. In Example 1, the error belongs to NS and NM with the certainty 0.75 and 1. Further, the trend belongs to MI with the certainty 1. In this case, two rules in Figure 5, i.e., $\{NS, MI\} \rightarrow DS$ and $\{NM, MI\} \rightarrow DM$, are relevant. Since the error and trend belong to NS and MI with the certainty 0.75 and 1, the certainty of the first rule $\{NS, MI\} \rightarrow DS$ is: min{0.75,1} = 0.75. At the same time, the error and trend belong to NM and MI with the certainty 0.25 and 1; therefore, the certainty of the second rule $\{NM, MI\} \rightarrow DM$ is: min{0.25,1} = 0.25.

Finally, the defuzzification interface derives crisp $d'(k)$ from the rules. For defuzzification, QCM and SCM use the MFs in Figure 7(c): Drop None, Drop Few, Drop Some, and Drop Most. To describe the defuzzification process, let x and y represent the row and column indexes in Figure 5 ($1 \leq x \leq 7$ and $1 \leq y \leq 6$). Also, let $\mu(x, y)$ denote the certainty of the corresponding $rule(x, y)$ in the table. Further, let $c(x, y)$ denote the center of the fuzzy set that is the $rule(x, y)$'s

[5] If $e(k)$ or $\Delta s(k)$ is smaller than -0.5 (larger than 0.5), it belongs to NL (PL) with certainty 1. Similarly, if $f(k) < -3.5$ ($f(k) > 3.5$), it belongs to SD (SI) with certainty 1.



(a) MFs used to fuzzify $e(k)$ or $\Delta s(k)$

(b) MFs used to fuzzify $f(k)$

(c) MFs used to derive $d'(k)$ and $d(k)$

Fig. 7. Membership Functions (MFs) for Fuzzification and Defuzzification

consequent. For a triangular fuzzy set, the center is the middle of the triangle's base. In this paper, we use the centroid method [9] to compute the drop probability for per-stream queue length control:

$$d'(k) = \frac{\sum_{x,y} \mu(x,y) \cdot c(x,y)}{\sum_{x,y} \mu(x,y)} \quad (7)$$

Example 3. Defuzzification. In Example 2, two control signals DS and DM are derived with certainty values 0.75 and 0.25, respectively. According to Figure 7(c), the center of DS is 0.475 and the center of DM is 0.712. Using the certainty values of the rules from the previous example, we compute the drop probability: $d'(k) = \frac{0.75 \cdot 0.475 + 0.25 \cdot 0.712}{0.75+0.25} = 0.534$. Thus, this stream's tuples will be randomly dropped with probability 0.534 until the next control event happens, unless SCM further adjusts the drop probability based on the selectivity change $\Delta s(k)$.

IV. Performance Evaluation

In this section, a description of the experimental settings is followed by the performance evaluation results.

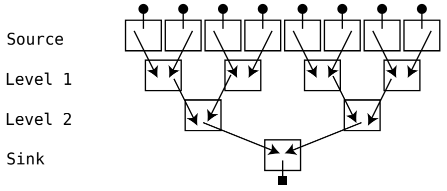


Fig. 8. Query network topology for experiments

For our experiments, we use CAIDA's Internet traffic traces collected by the *equinix-sanjose* high-speed monitor [17]. The average bit-rate of the monitored network is around 2.3 Gbps. Approximately 500,000 packets on average are being processed every second by each of the micro-benchmark applications that we have used for performance evaluation. Thus, our applications model the challenging problem of monitoring bursty and highly dynamic Internet traffic patterns.

Figure 8 shows the query network topology used in our experiments. The input data streams flow from the sources and processed in the four layers to generate the query results at the sink. The 15 physical machines in the figure are connected via a 1 Gbps switch. Each machine has a 1.5 GHz or 2.2 GHz Intel dual core CPU and 1GB main memory. In our experiments, the stream sources use the faster machines to generate high-rate data streams. Input data are stored in raw packet format and partitioned to eight source level machines based on their destination addresses, modeling network monitoring sensors placed at different locations. Sources read blocks of the traces, transform the data into Borealis tuples, and inject them into the input streams of the query network. The 8 query chains in Figure 8 are executed in parallel. For each query operator, we use a jumping window of 1000 tuples. Thus, an individual stream query operator is executed for every 1000 tuple arrivals.

Table I shows the flow statistics (FS) application. In FS, the flow information, including source and destination addresses as well as port numbers, is extracted from the TCP/IP packets to continuously analyze flow statistics for the source-destination pairs found in the jumping window. (For brevity, the window constraint is not included in the table.) The continuous queries in the table are deployed at the specified levels in Figure 8 to analyze network traffic patterns based on flow IDs. The sink node computes a running average and reports flows whose rates are higher than the value specified in Table I. Although we have done experiments for other network analysis applications, we mainly report the results of FS due to space limitations.

In this paper, we compare the performance of our approach, called FuzzyQS, to that of Borealis, which employs FIT [16] for load shedding, mainly in terms of the queue length and CQ processing delay. At every second, we record the maximum queue length measured in the query network. In this paper, we set $Q_S = 1000$ tuples for performance evaluation purposes. For performance evaluation, we keep track of the biggest queue size in the whole distributed DSMS at every second. In addition, the largest source-to-sink query processing delay is measured every 1s. Since our approach avoids overloads via load shedding on a per-stream and per-operator basis, they can indirectly reduce the delay too.

Figure 9 shows the *cumulative* distribution of the queue length for 10 experimental runs of the FS application. As shown in the figure, the 95 percentile queue length of Borealis and FuzzyQS is 59435 and 857 tuples, respectively. Figure 10 shows the *transient* queue length measured every 1s for 1 run. FuzzyQS controls the per-stream queue length to be below Q_S = 1000 tuples for most of the time except for a few

DSMS Application	Level	Query
Flow Summary (FS)	Source	**select** * **from** (**select** count(*), sum(size),flowid **from** (**select** [flow fields] **from** input) **group by** flowid) **where** count > 5
	Level 1	**select** * **from** (**select** sum(count), sum(size), flowid **from** (**select** * **from** input1,input2) **group by** flowid) **where** count > 10
	Level 2	**select** * **from** (**select** sum(count), sum(size), flowid **from** (**select** * **from** input1,input2) **group by** flowid) **where** count > 20
	Sink	**select** * **from** (**select** sum(count), sum(size), flowid **from** (**select** * **from** input1,input2) **group by** flowid) **where** size > avg(size) / 2

TABLE I
FLOW ANALYSIS APPLICATION DEPLOYED OVER THE QUERY NETWORK IN FIGURE 8

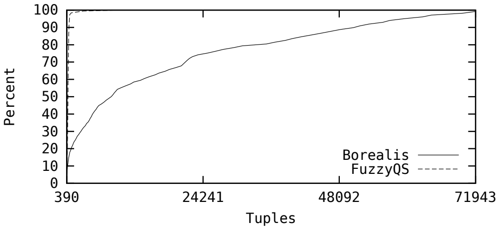


Fig. 9. Flow Summary: Cumulative Queue Length for 10 Runs

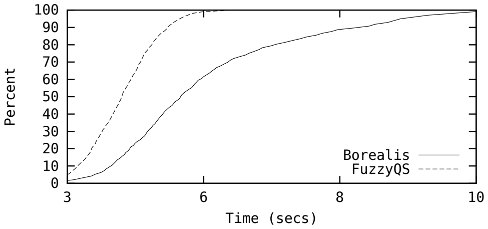


Fig. 11. Flow Summary: Cumulative Delay for 10 Runs

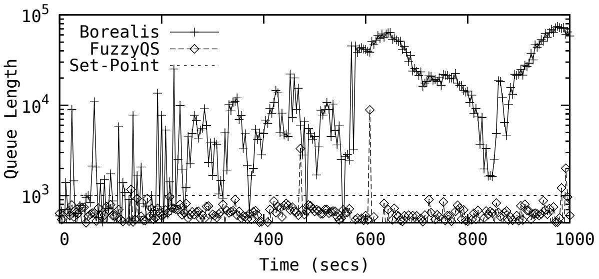


Fig. 10. Flow Summary: Transient Queue Length for 1 Run

overshoots as shown in Figure 10. From Figures 9 and 10, we observe that the per-stream queue length of FuzzyQS is up to two orders of magnitude shorter than that of Borealis. By controlling the per-stream backlog, FuzzyQS considerably decreases the delay too. In Figure 11, the *95 percentile delay* of Borealis and FuzzyQS for 10 runs is 8.86s and 5.26s, respectively. Thus, the 95 percentile delay of our approach is approximately 40% shorter than that of Borealis. FuzzyQS achieves these improvements via per-stream, fully-distributed queue length control based on fuzzy rules. Borealis makes system-wide decisions on load shedding at the nodes near the sinks. The decisions are aggregated and cascaded back toward the sources in an incremental manner [16]. Notably, existing DSMS, including [1], [2], [3], [5], [4], [6], [18], [19], [13], [14], [15], [8], [16], do not aim to support the desired per-stream queue length even in the presence of dynamic workloads. In the future, we will investigate how to further reduce per-stream queue length overshoots.

Figure 12 shows the cumulative output rate for 10 runs, which measures the number of the output tuples leaving the sink per second. From the figure, we observe that the output rate of FuzzyQS is similar to the rate of Borealis. Thus, our approach does not simply drop more tuples than Borealis does to control the queue length. Instead, it judiciously adapts the drop rate of each stream, if necessary, to avoid forming a bottleneck in the query chains.

In addition to FS, we have designed and run two more applications to analyze network traffic patterns based on the source and destination IP address, respectively. In these experiments, FuzzyQS has closely supported Q_s. The queue length of Borealis is up to an order of magnitude longer that that of FuzzyQS. The response time of Borealis is longer than ours by 0.63s - 1.2s. Also, FuzzyQS has shown simliar output rates to those provided by Borealis. Detailed results for these DSMS applications are not included due to space limitations.

Moreover, our approach incurs *little overhead*. In our experiments, our load shedding scheme is activated approximately 80 times/s on average, consuming approximately 0.25% total CPU utilization. It consumes less than 1KB of memory in each node to store the two rule-bases and keep track of the queue lengths and operator selectivities.

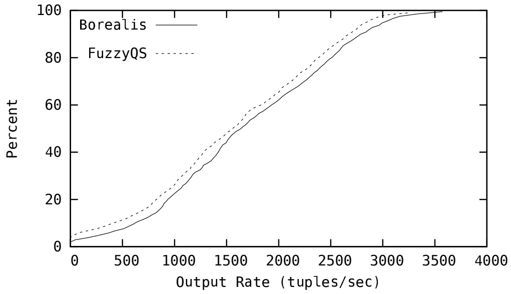


Fig. 12. Flow Summary: Cumulative Output Rates for 10 Runs

V. Related Work

Load shedding techniques have been studied to enhance the performance and reliability of DSMS [13], [14], [8], [16]. Under overload, the load shedder of Aurora drops the tuple with the lowest utility first under the assumption that the utilities of streams are known *a priori* [8]. In Borealis, global load shedding decisions are made near the sink and propagated back toward the stream sources [16]. Data Triage does load shedding under overload, while generating short summaries of the discarded packets [13]. This approach is complementary to ours in that it could be combined with our approach to support a summary of dropped tuples in the background. Overall, our work is different in that it supports fully localized, per-stream backlog management by leveraging continuous query semantics, while applying well-established fuzzy control techniques.

A proportional and integral (PI) controller is designed to bound the average per-tuple delay via load shedding [14]. It has also been used to support real-time periodic queries [18], [19], [23] implemented in STREAM [5]. However, a PI controller may largely fail to support the desired performance, if the model of the controlled system, e.g., a DSMS, derived offline using representative workloads becomes inaccurate due to dynamic workloads and system behaviors [10]. Unfortunately, finding representative workloads for a highly-dynamic, nonlinear systems, such as DSMS, is very hard (if at all possible). Applying other branches of control theoretic techniques based on mathematical modeling of the controlled system, such as [11], [12], also suffers from the difficulty of accurately modeling highly-dynamic and nonlinear DSMS.

Fuzzy logic control has been applied to manage the performance of other computational systems such as real-time tracking [24]. Basaran et al. [25] show that fuzzy control considerably outperforms a PI controller and a model predictive controller in terms of CPU utilization control in a real-time operating system. However, these approaches do not aim to support load shedding considering distributed DSMS semantics discussed in Section I. Nor do they support event-driven control to manage the per-stream backlog.

VI. Conclusion and Future Work

In this paper, we present a lightweight load shedding scheme to manage the per-stream backlog. Our approach directly considers the per-stream backlog and selectivity of each query operator to support graceful, fine-grained load shedding in a distributed manner. Further, our approach is event-driven. It quickly reacts to bursty workloads, while avoiding an unnecessary activation of the controller under light load conditions. The performance of our approach is evaluated thoroughly using high-rate Internet traces by extending Borealis, which is an advanced open-source distributed DSMS. From our experiments, we observe that our approach can closely support the desired per-stream queue length, while considerably decreasing the per-stream backlog and query processing delay compared to Borealis. In the future, we will explore more effective scheduling and load shedding schemes to further enhance the performance of distributed DSMS.

Acknowledgement

This work was supported, in part, by the NSF grant CNS-117352 and supplementary project to AFOSR grant FA9550-09-1-0165.

References

[1] "StreamBase." [Online]. Available: http://www.streambase.com/

[2] "InfoSphere Streams." [Online]. Available: http://www-01.ibm.com/software/data/infosphere/streams/

[3] "Microsoft StreamInsight." [Online]. Available: http://msdn.microsoft.com/en-us/library/ee362541.aspx

[4] D. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *CIDR*, 2005.

[5] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query Processing, Approximation, and Resource Management in a Data Stream Management System," in *CIDR*, 2003.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "TelegraphCQ: Continuous Dataflow Processing," in *SIGMOD*, 2003.

[7] N. Tatbul, "Load shedding techniques for data stream management systems," Ph.D. dissertation, Brown University, 2007.

[8] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load Shedding in a Data Stream Manager," in *VLDB*, 2003.

[9] W. Pedrycz, *Fuzzy Control and Fuzzy Systems*. Wiley and Sons, Inc., 1993.

[10] C. L. Phillips and H. T. Nagle, *Digital Control System Analysis and Design*. Prentice Hall, 1995.

[11] K. J. A. ström and B. Wittenmark, *Adaptive Control*. Addision Wesley, 1995.

[12] J. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.

[13] F. Reiss, "Declarative Network Monitoring with an Underprovisioned Query Processor," in *ICDE*, 2006.

[14] Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao, "Load Shedding in Stream Databases: A Control-Based Approach," in *VLDB*, 2006, pp. 787–798.

[15] B. Mozafari and C. Zaniolo, "Optimal Load Shedding with Aggregates and Mining Queries," in *ICDE*, 2010.

[16] N. Tatbul, U. Çetintemel, and S. B. Zdonik, "Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing," in *VLDB*, 2007.

[17] K. C. Claffy, D. Andersen, and P. Hick. The CAIDA Anonymized 2012 Internet Traces. [Online]. Available: http://www.caida.org/data/passive/passive_2012_dataset.xml

[18] Y. Wei, S. H. Son, and J. A. Stankovic, "RTSTREAM: Real-Time Query Processing for Data Streams," in *ISORC*, 2006.

[19] K. Kapitanova, Y. Wei, W. Kang, and S. H. Son, "Applying Formal Methods to Modeling and Analysis of Real-time Data Streams," *Journal of Computing Science and Engineering*, vol. 5, no. 1, 2011.

[20] "Linear Road: A Stream Data Management Benchmark," http://pages.cs.brandeis.edu/ linearroad/.

[21] "1998 World Cup Web Site Access Logs," http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[22] F. Reiss, "Data Triage: An Adaptive Architecture for Load Shedding in TelegraphCQ," in *ICDE*, 2005.

[23] Y. Zhang, C. Huang, and D. Zhang, "A Novel Adaptive Load Shedding Scheme for Data Stream Processing," in *Future Generation Communication and Networking*, vol. 1, 2007.

[24] B. Li and K. Nahrstedt, "A Control-Based Middleware Framework for Quality of Service Adaptations," *Journal on Selected Areas in Communications*, vol. 17, no. 9, 1999.

[25] C. Basaran, M. H. Suzer, K. D. Kang, and X. Liu, "Robust Fuzzy CPU Utilization Control for Dynamic Workloads," *Journal of Systems and Software*, vol. 83, no. 7, pp. 1192–1204, 2010.

Grex: An Efficient MapReduce Framework for Graphics Processing Units

Can Basaran, Kyoung-Don Kang

Department of Computer Science, Binghamton University

Abstract

In this paper, we present a new MapReduce framework, called Grex, designed to leverage general purpose graphics processing units (GPUs) for parallel data processing. Grex provides several new features. First, it supports a parallel split method to tokenize input data of variable sizes, such as words in e-books or URLs in web documents, in parallel using GPU threads. Second, Grex evenly distributes data to map/reduce tasks to avoid data partitioning skews. In addition, Grex provides a new memory management scheme to enhance the performance by exploiting the GPU memory hierarchy. Notably, all these capabilities are supported via careful system design without requiring any locks or atomic operations for thread synchronization. The experimental results show that our system is up to 12.4x and 4.1x faster than two state-of-the-art GPU-based MapReduce frameworks for the tested applications.

1. Introduction

As the modern GPUs are becoming more programmable and flexible, their use for general purpose computation is increasing. NVIDIA's CUDA [1] and ATI's OpenCL [2] support general purpose GPU programming. Generally, a GPU has a significantly larger number of processing elements than a modern multicore CPU does, providing massive hardware parallelism. For example, an NVIDIA GeForce 8800 GPU provides 128 cores and a maximum 768 resident threads. Also, a more advanced NVIDIA Fermi GPU provides 512 cores that can accommodate up to 1536 threads.[1] Especially, GPUs are designed to support immense data parallelism; that is, GPU hardware is optimized for threads to independently process different chunks of data in parallel.

MapReduce [3] is a popular programming model designed to process a large amount of data in parallel. A user developing a data parallel application via

Email addresses: `cbasaran@cs.binghamton.edu` (Can Basaran), `kang@binghamton.edu` (Kyoung-Don Kang)

[1] In this paper, we focus on NVIDIA GPUs; however, the principal ideas of our approach are generally applicable to the GPUs from the other vendors. Hereafter, a GPU refers to an NVIDIA GPU.

Preprint submitted to Elsevier *September 20, 2012*

MapReduce is only required to write two primitive operations, namely map and reduce functions. A MapReduce application is called a job. Given a job, the MapReduce runtime partitions the job into smaller units called tasks. It assigns tasks and data chunks to workers, which have available computational resources, called slots, to process the data chunks by running the tasks.[2] The workers process different data chunks in parallel independently from each other. The workers running map tasks process input <*key, value*> pairs to generate intermediate <*key, value*> pairs. The reduce function merges all intermediate <*key, value*> pairs associated with the keys that are in the same logical group. As the runtime handles scheduling and fault tolerance in a transparent manner, it is easy for a user to write data parallel applications using MapReduce.

Given that both GPUs and MapReduce are designed to support vast data parallelism, it is natural to attempt to leverage the hardware parallelism provided by GPUs for MapReduce. Although doing this may seem simple on the surface, it faces a number of challenges especially to process variable size data for industrially relevant and highly data parallel applications, such as natural language processing and web document analysis:

- Most GPUs do not support dynamic memory management. Thus, it is challenging to process variable size data in GPUs. Furthermore, the maximum size of variable length data is often unknown *a priori* in natural language processing and web document analysis. Hence, existing techniques developed to deal with variable size data, such as padding, are not directly applicable.[3]

- While processing variable size data, MapReduce tasks may suffer from load imbalance. To process variable size textual data, the MapReduce runtime sequentially scans the entire input data and assigns the same number of lines to each map task in a phase called input split. Unfortunately, input split is not only slow due to the sequential execution but also subject to potential load imbalance among the map tasks, since the amount of data may vary from line to line.

- In the reduce phase of MapReduce, an entire group of intermediate data having the same key is hashed to a single reduce task, incurring severe load imbalance among the reduce tasks in the presence of data skews [5, 6]. For example, certain words or URLs tend to appear more frequently than the others in natural language or web documents. As a result, a reduce task assigned a popular key group is likely to have a significantly larger number of intermediate <*key, value*> pairs to process than the other reducers do. Load imbalance due to the input split and data skews may adversely affect

[2] In Hadoop [4], an open source implementation of MapReduce by Yahoo, a slot represents an available CPU core by default.

[3] To apply padding, the entire set of the variable size data has to be preprocessed to find the size of the biggest data item first. After that, actual padding needs to be performed. These extra steps may introduce a significant overhead when the data set is large.

the performance, since it is known that the slowest task determines the speed of a job that consists of multiple tasks executed in parallel.

- GPU memory hierarchy is significantly different from the host (i.e., CPU side) memory hierarchy. Since the number of the cycles consumed by the fastes and slowest memory in the hierarchy is different by several orders of magnitude, it is challenging but important to efficiently utilize the GPU memory hierarchy.

To tackle these challenges, we design, implement, and evaluate a new GPU-based MapReduce framework called Grex (meaning a herd in Greek). Our goal is to efficiently process variable size data for the entire MapReduce phases beginning from input split to reduction, while minimizing potential load imbalance in MapReduce. At the same time, we aim to efficiently utilize the GPU memory hierarchy, which is important for general MapReduce applications executed in GPUs to process either variable or fixed size data. A summary of the key contributions made by Grex towards achieving these goals follows:

- To efficiently handle variable size data, Grex replaces the sequential split phase in MapReduce with a new *parallel split* step. In parallel split, Grex *evenly distributes input data* in terms of the number of bytes to the threads that tokenize input data in parallel.[4] Notably, Grex does not need to know the size of any token in advance to do parallel split.

- After parallel split, each thread executes the user-specified map() function to process the tokens found as a result of parallel split. After finishing to execute the map() function, each thread that has found the beginning of a token computes a *unique address* to which an intermediate data item will be written without communicating with the other threads. Therefore, *no locks or atomics* for synchronization are needed. Similarly, the output of the reduce phase is written without requiring any locks or atomic operations.

- Grex assigns an *equal number of intermediate <key, value> pairs to the reduce tasks* to execute the user-specified reduce() function in a parallel, load-balanced manner even if the keys are skewed.

- Although MapReduce deals with <*key, value*> pairs, only keys are needed for a considerable part of data processing in a number of MapReduce applications. In word count, for example, keys can be processed by map tasks and sorted and grouped together with no values. After grouping the same keys together, the frequency of each unique key has to be computed.

[4] In this paper, a meaningful unit of variable size data, e.g., an English word or URL, is called a token. Converting raw data, e.g., natural language or web documents, into tokens is called tokenization. For fixed size input data such as integers or floats, Grex simply assigns the same number of data items to the map tasks and skips the parallel split step.

Thus, it is not necessary to generate values in an earlier stage in certain applications. Based on this observation, Grex supports a new feature called *lazy emit*. Grex provides an API to let a user specify the MapReduce phase of a specific application where Grex needs to create values and emit complete <*key, value*> pairs to the GPU DRAM, called global memory. Before the user-specified phase begins, Grex does not generate or store any values but only deals with keys. By doing this, Grex intends to decrease the overhead to handle intermediate data in terms of computation and memory usage.

- Through all the MapReduce phases, Grex takes advantage of the *GPU memory hierarchy* to decrease the delay for memory access. To this end, Grex provides an API by which a user (i.e., a MapReduce application developer) can specify the size and type (read-only or read/write) of a buffer that will be stored in global memory and cached by Grex using faster memory of the GPU in a transparent manner.

Although a number of GPU-based MapReduce systems have previously been developed [3, 4, 7, 8, 9, 10, 11, 12], we are not aware of any other GPU-based MapReduce framework that supports all these features.

We have implemented Grex and tested our implementation on a PC with an AMD quad-core CPU using two different generations of NVIDIA GPUs. We have compared the performance of Grex to that of Mars [7] and MapCG [8], which are open source GPU-based MapReduce frameworks. For performance evaluation, we have used six common applications with different needs in terms of MapReduce computations, i.e., string match, page view count, word count, inverted index, similarity score, and matrix multiplication. The first four applications deal with variable size data, while the last two applications process fixed size data. By considering a diverse set of applications, we thoroughly evaluate the performance of Grex in comparisons to Mars and MapCG. Our performance evaluation results indicate up to 12.4x speedup over Mars and up to 4.1x speedup over MapCG using the GeForce GTX 8800 GPU and GeForce GTX 460 GPU that supports atomic operations needed by MapCG, respectively.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. A brief description of MapReduce and the general GPU architecture is given in Section 3. The design of Grex is described in Section 4. Grex code snippets are discussed in Section 5 to further illustrate the key concepts introduced in this paper. The performance evaluation results are discussed in Section 6. Finally, Section 7 concludes the paper and discusses future work.

2. Related Work

MapReduce is developed by Google [3]. Hadoop [4] is an open-source implementation of MapReduce developed by Yahoo. Both MapReduce and Hadoop are designed for commodity servers. Phoenix [9] and its extensions [10, 13] are one of the first projects to implement a MapReduce framework for a multi-core architecture with shared memory. Ostrich [14] intends to improve Phoenix by implementing an iterative map phase. Similarly, Grex iteratively processes input data by applying the tiling technique [15]. There is also a MapReduce implementation for the IBM Cell architecture [16].

Property	Grex	Mars	MapCG	[11]	Mithra	GPMR
Single pass execution	yes	no	yes	yes	yes	yes
Locks or atomics	no	no	yes	yes	no	no
Parallel split	yes	no	no	no	no	no
Load balancing	yes	no	no	no	no	yes
Lazy emit	yes	no	no	no	no	no
GPU caching	yes	no	no	yes	no	no
GPU cluster	no	no	no	no	yes	yes

Table 1: Comparisons of the GPU-based MapReduce frameworks

Several GPU-based MapReduce frameworks have been developed by different researchers. A GPU-based MapReduce framework developed by Catanzaro et al. [17] supports a constrained MapReduce model that assumes that the keys are known in advance and a map function produces only one output. Due to these assumptions, the applicability of [17] is limited. Especially, it is not directly applicable to process raw (i.e., not tokenized) variable size data. Table 1 summarizes the key features provided by several GPU-based MapReduce frameworks that support full MapReduce functionalities. Mars [7] is a GPU-based MapReduce framework. It shows up to 10x speedup over Phoenix [9]; however, it is not optimized for the GPU architecture. To overcome the lack of dynamic memory management in many GPUs, Mars computes the precise amount of GPU memory required for processing a MapReduce job in the first pass. It does actual data modification and generation in the second pass. Since the first pass performs every MapReduce task except for producing actual output, Mars essentially executes a MapReduce job twice. Neither does it utilize the GPU memory hierarchy.

MapCG [8] uses atomic operations to support dynamic memory management in global memory. In this way, MapCG avoids two pass execution of Mars. However, atomic operations require sequential processing of dynamic memory allocation or deallocation requests, incurring performance penalties. Ji and Ma [11] enhance the performance of MapReduce applications using shared memory in a GPU. However, in their approach, half the threads are used to support synchronization based on atomic operations, incurring potentially non-trivial overheads. Different from these GPU-based MapReduce frameworks [17, 7, 8, 11], Grex 1) processes a MapReduce job in a single pass, 2) supports parallel split, 3) facilitates load-balanced data partitioning among the map and reduce tasks of a job, 4) supports lazy emit, and 5) efficiently utilizes the GPU memory hierarchy for performance enhancement. It takes advantage of not only shared memory but also constant memory and texture cache, while requiring no atomic operations or locks.

Mithra [18] uses Hadoop as a backbone to cluster GPU nodes for MapReduce applications. By building a GPU cluster on top of Hadoop, Mithra takes advantage of powerful features of Hadoop such as fault tolerance, while leveraging a large number of GPU threads. As a result, it significantly outperforms a CPU-only Hadoop cluster. However, it does not support the key optimizations of Grex described before. GPMR [12] is an open source MapReduce framework designed for a GPU cluster. It relies on simple clustering of GPU nodes without supporting a distributed file system or fault tolerance. Instead, it focuses on providing novel features for partial data aggregation to decrease the I/O between the CPU and GPU in a node as well as network I/O between the map and reduce tasks. However, GPMR does not consider to support the main features of Grex. Also, the current GPMR implementation processes already tokenized data only. Thus, variable size data have to be tokenized using the sequential input split phase in Hadoop or by any other means before being processed by GPMR. Mithra [18] and GPMR [12] are complementary to Grex in the sense that the current version of Grex uses a single GPU, similar to Mars [7], MapCG [8], and [11]. Grex could be combined with Hadoop or GPMR to support MapReduce applications in a GPU cluster. A thorough investigation is reserved for future work.

Kwon et al. [6] investigate several causes of skews in Hadoop applications. Grex evenly distributes data to the map and reduce tasks. This approach will alleviate partitioning skews; however, we do not claim Grex completely solves skew problems. For example, a certain <*key, value*> pair can be computationally more expensive to process than the others [6]. Efficiently dealing with potential skews in a GPU-based MapReduce framework is an important yet rarely explored research problem. Other GPU-based MapReduce frameworks including [7, 8, 11, 12, 18] do not consider this problem.

There have been many recent applications of GPU computing to different problem domains such as Fast Fourier Transform [19], matrix operations [20], encryption and decryption [21], genetics [22], intrusion detection [23], and databases [24]. Using GPUs, these research efforts have achieved significant performance improvement [25, 26]. These results call for a general GPU-based parallel programming framework to make the architectural advantages of GPUs available with reduced complexity compared to manual architectural optimizations.

3. Preliminaries

In this section, backgrounds of MapReduce and GPUs needed for the rest of the paper are given. Through the paper, word count is used as a running example to explain key concepts of MapReduce and Grex especially in terms of dealing with variable size data. Although word count is conceptually simple, it requires all the MapReduce phases and deals with variable-size textual data challenging to handle in a GPU.

3.1. MapReduce

A MapReduce framework executes an application as a set of sequential phases such as split, map, group, and reduce. Each phase is executed by a set of tasks that run in parallel. A task is sequentially processed by a single processing element called a worker. In the split phase, the input is split into smaller subproblems called records and distributed among the workers. For example, a text file such as a web document can be split at new lines in the word count application. In the map phase, the input data is mapped to intermediate <*key, value*> pairs by a user defined map() function. In word count, a map task can be designed to process a user-specified number of text lines to emit <*word, 1*> pairs for each word in the lines. In the group phase, intermediate <*key, value*> pairs are sorted and grouped based on their keys. Finally, the intermediate pairs are reduced to the final set of <*key, value*> pairs by a user defined reduce() function. As an example, for 10 <*'the', 1*> intermediate <*key, value*> pairs produced by the map tasks, one new pair, <*'the', 10*> is emitted as the final output. (Some applications, e.g., string match, do not need grouping or reduction.)

Generally, a user is required to implement the map() and reduce() functions and the grouping logic, which are often specific to applications. Since a map/reduce task or grouping is executed sequentially in MapReduce, a user is only required to specify *sequential* logic. The rest of the problem, such as data distribution to tasks, collection of results, and error recovery for parallel processing, is handled by the MapReduce runtime system. Thus, developing a parallel application via MapReduce does not impose undue complexity of parallel programming on users.

3.2. General Purpose Graphics Processing Units

Graphics processors have architectural advantages that can be harnessed to solve computational problems in domains beyond graphics [26]. Further, the introduction of general purpose programming languages such as CUDA [1], Stream [27], and OpenCL [2] has promoted their use as general purpose processors [26].

GPU code has an entry function called a *kernel*. A kernel can invoke other (non-kernel) device functions. Once a kernel is submitted for execution, the host has to wait for its completion. No interaction is possible with a running kernel. A GPU has a number of SIMT (Single Instruction Multiple Threads) multiprocessors as shown in Figure 1. Threads are temporarily bound to these SIMT elements. Thus, a GPU can accommodate thousands or even more live threads. The number of processing elements and maximum number of active threads vary among GPUs [26, 28]. Threads are grouped into blocks. A (thread) block is scheduled dynamically to run on one of the available multiprocessors. All the threads of one block execute the same code because the underlying architecture is SIMT. When the threads in a block branch to different code segments, branches are executed sequentially. This phenomenon, called control divergence, can result in dramatic performance degradation. Thus, the threads in Grex are designed to execute the same logic at a time to avoid divergence.

To support efficient memory access, GPUs employ a sophisticated memory hierarchy [29]. Specifically, there are five types of memory as shown in Figure 1: 1) a set of general propose registers per thread; 2) shared memory that is shared by all the threads in a block. It can be used as a software managed cache to cache data in the off-chip global memory; 3) a read-only constant memory that can be used as software managed cache for the global memory or to store frequently accessed constant data that is small in size; 4) read-only hardware managed texture cache that caches global memory; and 5) global memory. The size and access delay generally increases from 1) to 5). General purpose registers are several orders of magnitude faster than global memory whose access latency is hundreds of cycles. Global memory is the largest memory in the GPU. It can be accessed by any thread and even from the host CPU. However, it is the slowest memory.

Shared memory is almost as fast as the registers. Every thread of one thread block shares the shared memory; therefore, the threads in one thread block can communicate easily with each other. However, a thread in one thread block cannot access the shared memory of a different thread block. Constant and texture memory have short access latency upon a cache hit. Upon a miss, they have the same latency as global memory. Constant and texture memory can be used to store read-only data. Any thread can access any data stored in constant or texture memory. Since constant memory allocation is static, the amount of constant memory to be used must be known at compile time. As a result, constant data have to be copied to the fixed address before it can be accessed. Hence, in Grex, constant memory is used to keep fixed-size read-only data, such as a translation table used to convert upper case characters to lower case ones in word count. If the size of a user specified data structure exceeds the amount of constant memory, a compile time error is issued by Grex. Unlike shared memory and constant memory, texture cache is managed by hardware. It does not require static allocation and provides good performance for random

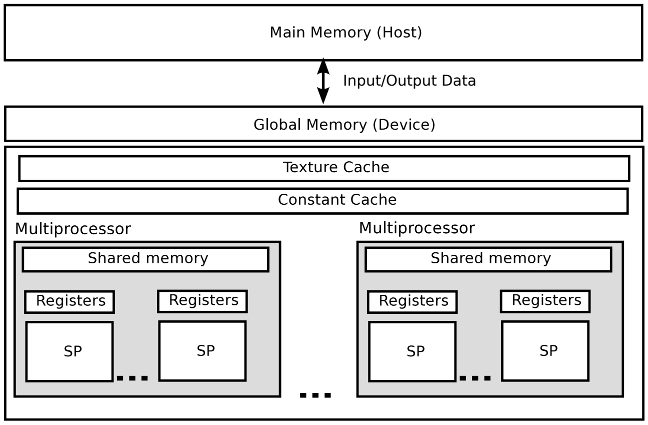


Figure 1: General Purpose GPU Architecture. (SP: Streaming Multi-Processor)

Algorithm 1 Control flow of Grex for processing variable size data

```
1. Read data blocks from global memory and cache them evenly in shared memory across the thread blocks. Execute the parallel split phase. (Skip this step if the data size is fixed.)
2. Execute the map phase. Compute a unique address for each intermediate key found after finishing the map phase. Using the unique addresses, write the keys to global memory in parallel without relying on locks or atomics.
3. Perform boundary merge in global memory to combine a key divided between two adjacent thread blocks, if any. (Skip this step if the data size is fixed.)
4. Group the intermediate keys via parallel sorting.
5. Read the intermediate keys from global memory and cache them evenly in shared memory across the thread blocks. Execute the reduce phase. Merge the results of the reduce phase performed by two adjacent thread blocks if necessary. (Skip this step if the data size is fixed.)
6. Perform lazy emit. Write the results to global memory. (Lazy emit is optional. If lazy emit is not used, the keys in the previous steps in this pseudo code should be replaced with <key, value> pairs.)
```

access of read-only data.

4. System Design

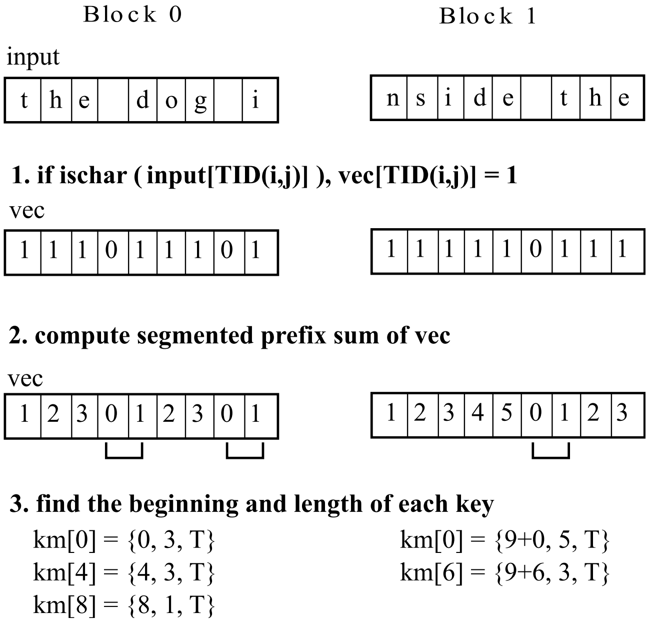
This section describes the MapReduce phases and buffer management scheme of Grex.

4.1. MapReduce Phases in Grex

Grex has five stages of execution: 1) parallel split, 2) map, 3) boundary merge, 4) group, and 5) reduce. A pseudo code that summarizes these stages is given in Algorithm 1. Note that the parallel split phase replaces the sequential input split phase of MapReduce [3, 4]. Also, lazy emit is not an additional phase but it postpones emitting values until the end of the entire MapReduce computation. Hence, only the boundary merge is added to the MapReduce control flow. Moreover, the parallel split and boundary merge phases are only needed to handle variable size data. If the size of data is fixed, Grex distributes the same number of data items, e.g., integers or floats, to each task for load balancing. Therefore, no parallel split or boundary merge is needed for fixed size data. A detailed description of the MapReduce phases of Grex follows.

1. Parallel split phase: In the parallel-split step, Grex initially assigns one unit of data to each thread. For example, it can assign a single character in the input documents to a thread to compute the starting position and length of each string in the input as shown in Figure 2. For variable size data, a token is a key. For example, a word or URL in word counting or web document analysis is a key. Thus, in the rest of this paper, we use keys and tokens interchangeably.

The j^{th} thread in the i^{th} thread block, $T_{i,j}$, is associated with a unique thread identifier: TID$(i,j) = i \times BS + j$ where BS is the size of the data unit handled by a thread block, which is equal to the number of the threads in a block in the word count example given in this paper. $T_{i,j}$ writes 1 (or 0) to `vec[TID(`i,j`)]` in Step 1 of Figure 2, if the assigned byte, i.e., input[TID(i,j)], is a token (or non-token) byte using the user-specified `istoken()` function such as `ischar()` in word count. (Bit sting `vec` is a temporary data structure allocated in shared memory.)



1. if ischar (input[TID(i,j)]), vec[TID(i,j)] = 1

2. compute segmented prefix sum of vec

3. find the beginning and length of each key

km[0] = {0, 3, T}
km[4] = {4, 3, T}
km[8] = {8, 1, T}

km[0] = {9+0, 5, T}
km[6] = {9+6, 3, T}

Figure 2: Kernel 1 - Parallel Split (Steps 1 and 2) and Map Phase (Step 3)

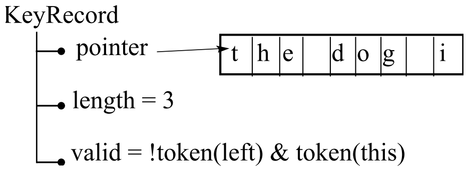


Figure 3: Key record data structure

Grex applies the parallel prefix sum algorithm [30] to the bit string to compute the starting position and length of each key in parallel as shown in Step

2 of Figure 2. To give an illustrative example of parallel split, we introduce a data structure in Figure 3, which shows a temporary data structure, called `keyRecord`, used for the parallel split and map phases. The pointer field points to the beginning of a variable size key, e.g., a word or URL. The length field is used to store the length of a key. Also, the valid attribute is used to indicate the beginning of a key. (This data structure is not necessary for fixed size data.)

An instance of `keyRecord` is referred to `km` in Figure 2 and stored in a register by a thread. As shown in Step 2 of Figure 2, thread $T_{i,j}$ knows that it has the beginning of a token, if $\texttt{vec[TID(}i, j-1\texttt{)]} = 0$ and $\texttt{vec[TID(}i, j\texttt{)]} = 1$ where $i \geq 0$ and $j \geq 1$ and sets $\texttt{km[TID(}i, j\texttt{)].valid} = \text{T(rue)}$ in Step 3 of Figure 2. It also sets the $\texttt{km[TID(}i, j\texttt{)].pointer}$ to the beginning of the key, while storing the length of the key to $\texttt{km[TID(}i, j\texttt{)].length}$.

In Grex, thread $T_{i,j}$ is assumed to have the beginning of a string, if $j = 0$ and $\texttt{vec[TID(}i, j\texttt{)]} = 1$. In fact, this is not always true, because a string can be divided between two adjacent blocks as shown in the figure. Such a case, if any, is handled by the boundary merge phase that will be discussed shortly. The boundary merge phase is needed, since two different GPU thread blocks cannot directly communicate with each other due to the current GPU hardware design.

The time complexity of the parallel prefix sum, which is the most expensive in the parallel split phase, is $\Theta((\frac{n}{N})lgn)$ where n is the length of the bit string indicating the input size in terms of the number of bytes and N is the total number of the GPU threads that can run at once. Thus, for large N, parallel split is considerably faster than a sequential algorithm used in MapReduce for tokenization, which is $\Theta(n)$. However, existing MapReduce frameworks including [3, 4, 7, 8, 9, 10, 11, 12] do not support parallel split.

Notably, a user only has to specify an `istoken()` function, such as the `ischar()` function in Figure 2. Parallel split is supported by the underlying Grex runtime system and hidden from the user. Since a user has to write a function similar to `istoken()` in any existing MapReduce framework for sequential input split, parallel split does not increase the complexity of developing a MapReduce application. Notably, all threads finish the parallel split phase simultaneously, because they process the same number of bytes using the same algorithm. Thus, no synchronization between them is required.

2. Map phase: A user specifies a `User::map()` function, similar to MapReduce [3] and Hadoop [4]. Grex further processes the keys in shared memory tokenized during the parallel split step via the map tasks, which execute the map() function in parallel. In word count, the parallel split and map phases are implemented as a single kernel, since the map function is only required to write the keys found in the parallel split phase to global memory. If `km[TID(i,j)].valid == T` in Step 3 of Figure 2, thread T_{ij} writes `km[TID(i,j)].pointer` into $\texttt{keys[}\lceil\texttt{TID(i,j)/2}\rceil\texttt{]}$ in global memory. Note that we divide the thread ID by 2, since a block cannot have more keys than half the bytes assigned to the

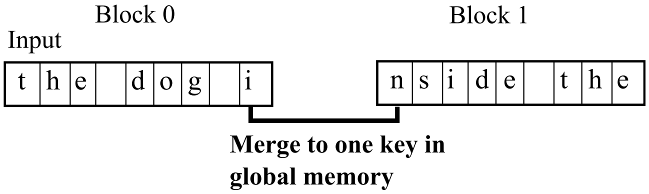


Figure 4: Kernel 2 - Boundary merge phase

block. A string, e.g., a word or URL, has at least one non-null character followed by a null character that indicates the end of the string. As the TID of a thread is unique, the threads that have valid pointers to the keys can write their pointers to the global memory location independently from each other. Thus, all the pointers to the beginnings of the keys are written to the global memory at the same time with no need to synchronize the threads.

The `input` data structure in Figure 2 is part of a Grex buffer, which keeps the input data and output of the map phase in global memory, respectively. Input is evenly divided among the thread blocks and cached in the shared memory of each block by Grex. Also, the `keys` data structure is the output buffer used to write the output of the map phase. A user can specify buffers needed to store input, intermediate, or output data in global memory. Grex caches them in shared memory or texture cache in a transparent manner. In addition, temporary data structures, such as `vec` and `km` in Figure 2, are created and managed by Grex. In the current version of Grex, `vec` and `km` are stored in shared memory and registers, respectively. After completing the execution of a kernel such as the one for parallel split and map, Grex deallocates the temporary data structures in shared memory.(Buffer management and caching is discussed in Section 4.2.)

In general, if the size of data is bigger than the shared memory size, the Grex steps are executed iteratively. Hence, the job size is not limited by the shared memory size. In addition, data upload to and download from the GPU in Grex is overlapped with the computation in the GPU to reduce the I/O overhead.

Grex supports an optional feature called **lazy-emit**, in which the value is neither created nor held in memory before the user-specified phase. For lazy-emit, a user only has to specify in which phase the value should be created and emitted to memory and what value should be emitted by defining a `User::emit(key)` function. For example, the emit() function for word count returns 1 for each word. Also, a user can optionally implement a `User::combine()` function to do partial reduction of intermediate <*key, value*> pairs produced in the map phase for an associative and commutative reduce() function, similar to MapReduce.

3. Boundary merge phase: We have designed a simple yet effective parallel method, called boundary merge, to check whether any key is split between neighboring thread blocks and merge it to a single key. If there are $B > 1$ blocks, Grex executes a kernel using $B - 1$ threads to do boundary merge in

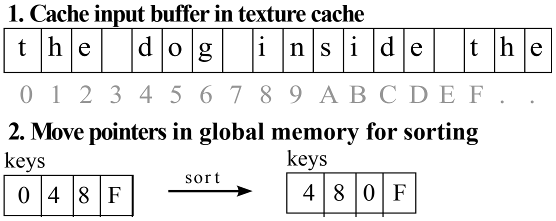

1. Cache input buffer in texture cache
t h e d o g i n s i d e t h e
0 1 2 3 4 5 6 7 8 9 A B C D E F . .
2. Move pointers in global memory for sorting
keys
0 4 8 F
sort
keys
4 8 0 F

Figure 5: Kernel 3 - Sorting intermediate keys

global memory, since there are $B - 1$ block boundaries. When the map phase is completed, the result is written to global memory as discussed before. The $B-1$ threads for boundary merge can start immediately when the kernel for the parallel split and map phases completes. Also, they do not have to coordinate or communicate with each other, since they do not share any data or other resources with each other. Thus, they work independently in parallel.

To do boundary merge, thread i ($0 \leq i < B - 1$) observes the end of the i^{th} block and the beginning of the $(i + 1)^{th}$ block to check whether or not both of them are valid (i.e., istoken() = true) as shown in Figure 4. If this condition is satisfied, thread i merges them to a single key and eliminates the partial key in the $(i + 1)^{th}$ block. Also, the thread decrements the number of the keys in the $(i+1)^{th}$ block. At the end of the boundary merge step, the pointers to the variable size tokens are stored in `keys` buffer in global memory as shown at the bottom left in Figure 5.

4. Grouping phase: In this step, the intermediate <*key, value*> pairs are sorted, similar to MapReduce and Hadoop. In this paper, parallel bitonic sorting is used for grouping. For sorting, a user only has to specify how to compare keys through the `User::cmp()` function. For example, a user can use the `strcmp()` function in word count.

To decrease the memory access delay for parallel sorting, we bind the input buffer in global memory to the on-chip texture cache as shown in Step 1 of Figure 5. We use the texture cache rather than shared memory, since sorting generally involves data accesses across thread blocks. For grouping in Grex, data comparisons are done in the texture cache, which is read-only and managed by the hardware. Actual movements of the pointers to variable size data, if necessary for sorting, are performed by Grex in global memory as shown in Step 2 of Figure 5. As a result, the pointers in the figure are sorted in non-ascending lexicographic order of the keys, i.e., words in word count.

For grouping, Mars [7] and MapCG [8] create an index space to keep the <*key, value*> pairs and apply hashing to find <*key, value*> pairs in global memory. Our approach is more efficient than these methods, since it reduces the memory consumption as well as the delay and contention for global memory access by leveraging texture cache and lazy emit, while requiring no index space.

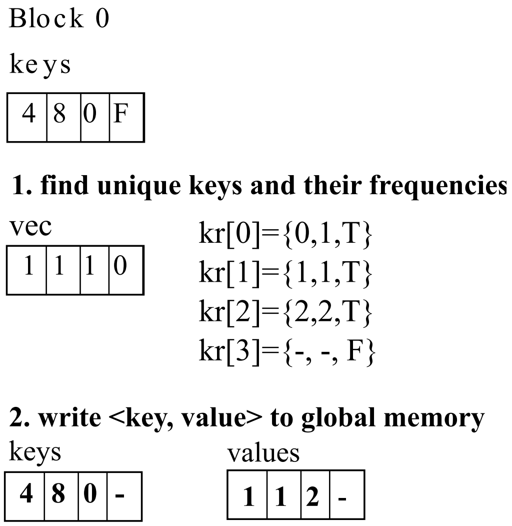


Figure 6: Kernel 4 - Reduction

5. Reduce phase: Grex provides a `User::reduce()` API to let a user write a reduce function specific to the application of interest. To significantly enhance performance, Grex uses a large number of thread blocks to reduce <*key, value*> pairs in a parallel, load balanced manner. Grex evenly distributes intermediate <*key, value*> pairs to each reduce task to avoid load imbalance due to data skews as discussed before. (Only one thread block is used in Figure 6 due to the small data size.)

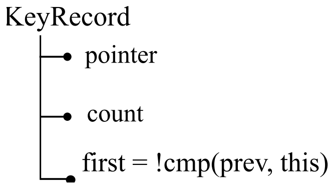


Figure 7: Key record data structure for reduction

In the reduce phase, Grex uses a temporary data structure in Figure 7. The pointer field points to the beginning of a variable size key. The count field is used to compute the frequency of a key's occurrences in natural language or web documents. Also, the first boolean attribute is used to indicate the first occurrence of a key. An instance of this data structure is called `kr`. It is stored in a register by a thread. (This data structure is unnecessary for fixed size data, similar to the temporary data structures used for the parallel split and map phases.)

In Step 1 of Figure 6, each thread is assigned a word and it compares its word to the word of the next thread (if any). A thread sets its bit in the temporary bit string, `vec`, newly allocated in shared memory, if the two strings are different. Using the bit string, the threads compute the exclusive parallel prefix sum to calculate the total number of *unique* keys. In Step 2, thread T_{ij} sets `kr[TID(i,j)].valid = T(rue)`, if it has a pointer to the first appearance of a key or vice versa. Also, `kr[TID(i,j)].pointer` and `kr[TID(i,j)].length` are set to the beginning of a unique key and the number of the occurrences, i.e., word count, of the key, respectively.[5] Finally, each thread writes a pointer to a variable size key, e.g., a word, to the `keys` buffer and a value, e.g., word count, to the `values` buffer in global memory using the information in the temporary `kr` data structure in Figure 6. Thus, the final output is <dog, 1>, <inside, 1>, and <the, 2> in this example.

Grex also applies a variant of the boundary merge technique in the reduce phase to produce correct final <*key, value*> pairs in global memory, if one group of <*key, value*> pairs with the same key is divided over more than one thread blocks due to load balancing among the reducers. An example code snippet of boundary merge is given in Section 5.

4.2. Buffer Management

In Grex, a user can use buffers managed by the Grex runtime system for efficient memory access in the GPU. A buffer in Grex is a data container that can partially or entirely reside in global memory, texture cache, or shared memory depending on the size and type of a buffer and the availability of shared memory or texture cache space. Grex provides a number of buffering methods and user APIs to enhance the performance by leveraging the GPU memory hierarchy.

To use a buffer, a user needs to specify the access mode and size of a buffer. Grex provides two buffer types: 1) a *constant buffer* and 2) a *buffer*. A constant buffer is stored in the constant memory. Its size should be fixed and known at compile time as discussed before. We describe non-constant buffers in the remainder of this subsection. A (non-constant) buffer is either read-only or read-write. If the user specifies a buffer as read-only, Grex uses the texture cache to cache the buffer. As the texture cache is managed by the hardware, a user does not need to specify the size. Grex simply binds a read-only buffer to the texture cache. Thus, data are loaded from the texture cache to shared memory by the hardware, if the data are available in the texture cache. On the other hand, if a buffer is read-write, Grex directly loads data from global memory to shared memory instead of checking the texture cache first. In Grex, a user can also specify a specific buffer to become active during a specific MapReduce phase via the `User::init()` callback function invoked and executed by Grex before processing each phase.

To process I bytes of input data, a user needs to specify the size of the buffer-/block, P, and the number of bytes processed by each GPU thread, $D(\geq 1)$.

[5] In the reduce phase, the `length` field is overloaded to indicate the word count. However, Grex is not tied to this overloading. Depending on a specific application of interest, a temporary data structure different from `keyRecord` can be used.

Algorithm 2 Grex buffer management

The following steps are executed in the CPU:

1. Invoke `User::init()` function. In this function the user enables/disables buffers for the current phase and modifies their properties as needed.
2. Before starting to execute the first phase of a job, upload input data to global memory.
3. Traverse the list of buffers to compute the total size of the shared memory cache needed by the enabled buffers. Also, bind buffers marked as read-only to the texture cache.
4. Launch the next MapReduce kernel with the calculated total cache size.

The following steps are executed in the GPU:

5. Initialize the shared memory cache assigned to each buffer in parallel.
6. Run the user-specified phase kernel.
7. Write the result of the kernel execution temporarily stored in in shared memory, if any, to the corresponding read-write buffer in global memory. (To do this, each thread that has data to write to global memory independently computes a unique address as discussed in Section 4.1.)

Thus, to process I bytes, Grex uses $B = \frac{I}{PD}$ blocks and $T = \frac{P}{D}$ threads/block. By default, the size of the shared memory used to cache a fraction of a buffer in each block is equal to the size of the per-block buffer in global memory. For example, if the size of a per-block buffer is 256 bytes, 256 bytes of shared memory in each thread block (if available) is used to cache the buffer by default. Grex also provides an option by which a user can specify a different amount of shared memory to be used to cache a buffer in each thread block.

In Grex, data are fetched to shared memory in sequence and the oldest data is replaced, if necessary, to accommodate new data, since a considerable fraction of MapReduce applications linearly scan input data to group and aggregate them in parallel. The overall workflow of the buffer manager in Grex is given in Algorithm 2. Grex buffer manager first calculates the total cache size needed by each thread block by traversing the user defined buffers. After computing the total size of the (shared memory) caches needed per thread block, Grex launches the kernel to execute a MapReduce phase in the GPU.

At the beginning of a phase, contiguous regions of shared memory in a thread block is assigned to the buffers active in the phase and initialized as specified by the user. Grex provides parallel cache initialization policies a user can choose for a cache: 1) The default policy reads data from global memory into the cache; 2) An alternative policy initializes every cache element to a constant value specified by the user; 3) The no initialization policy leaves the cache (e.g., a cache associated with an output buffer) uninitialized; and 4) The user value policy initializes each cache element to the return value of user defined `User::cache()` function that can be used to convert the input data. The selected policy is enforced by Step 5 of Algorithm 2.

After initializing caches, the kernel for the corresponding MapReduce phase is executed in Step 6 of Algorithm 2. After finishing the execution of a phase kernel, Grex writes the resulting data in shared memory back to global memory and deallocates shared memory. In this way, we aim to ensure that the result of executing a phase is globally available to the following MapReduce phases, if any. This approach allows the next MapReduce phase to fully utilize shared memory too. Since most MapReduce applications performs group-by and aggregation functions in parallel, the amount of results to be written back to global memory is not likely to be bigger than the original input data size. Thus, the overhead of write-back to global memory in Step 7 is expected to be tolerable. Moreover, Grex supports lazy emit to further decrease the shared memory consumption and the amount of data to write back to global memory.

In summary, Grex is designed to support the parallel split and load-balanced map and reduce phases, while efficiently utilizing shared memory and texture cache. In addition, Grex supports lazy emit and it is free of locks and atomics different from the other GPU-based MapReduce frameworks [7, 8, 11, 12, 18].

In the future, GPUs may support a unified cache model. However, even in a single cache model, it is important to carefully manage the cache. Our buffer management scheme may become simpler and more efficient in such an architecture, because it is unnecessary to use different caches based on read/write or read-only nature of a buffer. Also, if future GPUs allow a thread to directly read data cached by a different thread block, boundary merge can be performed not in the slow GPU DRAM but in the cache. Thus, it is possible for the performance of Grex to further improve in such an environment. In addition, the algorithmic approaches of Grex such as parallel split, even distribution of data to the mappers and reducers, and late emit will be still applicable regardless of the cache model.

5. Sample Grex Code

This section illustrate an example Grex code for word count to provide details on handling variable size data. The basic structure of word count is applicable to various text processing applications that require word or token extraction. A variable data application example is preferred since a fixed size data processing implementation is relatively trivial.

In Code 1, Grex calls the user specified `User::main()` function to create buffers and do system initialization. In this example, three (non-constant) buffers and one constant buffer are created. A user can use the `new_buffer <data_type> (per_block_size)` function call to create a buffer with a certain data type and size per thread block. Grex creates a buffer in global memory whose size is equal to the product of the per-block buffer size and the number of the thread blocks as discussed in Section 4. Also, it caches the buffer in shared memory of each thread block when the buffer is accessed at runtime. To create a constant buffer, a user has to use the `new_const_buffer <data_type> (total_size)` API as shown in the code. As a result of the Grex system call, one system-wide constant buffer of `total_size` is created in constant memory.

```
int Grex::User::main () {
  // Create a per-block buffer that contains 256 bytes of input data.
  input  = new_buffer<char>(256);
  // There are at most 256/2 words from 256 bytes of data due to the
  // null char at the end of a string.
  keys   = new_buffer<char*>(128);
  // Max no. of values is equal to max no. of words.
  values  = new_buffer<int>(128);
  // Create a constant buffer to hold the upper-case to lower-case
  // translation table
  tr_map  = new_cbuffer<char>(tr_table, 256);
  // Store boundary key sizes for merging. There can be at most 2
  // boundary words in a thread block.
  keysize = new_buffer<char>(2);
  ...
}
```

Code Snippet 1: Buffer setup

When Grex completes executing the main function, it reads the input data from disk as shown in Code 2 before it starts running a MapReduce job in the GPU. Data read from the input file are downloaded to global memory over the PCIe bus that connects the host and GPU. The data is read and the job is iteratively executed until the `read_input()` function returns the End of File (EOF). In this way, the GPU memory can be allocated and the job can be executed in an incremental manner. Grex also overlaps the I/O across the PCIe bus with the computation in the GPU, if any, to hide the I/O overhead.

```
size_t Grex::User::data(
  Buffer *b, char *d, off_t offset, size_t size){
  if(buffer->id()==input->id()) {
    input_file.seek(offset);
    return input_file.read(d, size);
  }
  return 0;
}
```

Code Snippet 2: Reading Input Data

Code 3 shows an example user initialization routine for the word count application. Grex invokes the `User::init()` function before starting to execute each phase kernel. In the function, the user needs to declare buffers that will be used in a specific phase and set the type of each buffer as either a read-write or read-only buffer. A buffer becomes usable once it is enabled and remains valid until it is explicitly disabled by the user.

When a buffer is enabled, it is set as a read-write buffer by default. A user needs to explicitly set a buffer as read only via the `read_only()` member function as shown in Code 3. A read-only buffer is cached in texture memory as discussed before.

Notably, there is no case statement for parallel split or boundary merge in Code 3, because the execution of the parallel-split and boundary merge phases plus the buffer management in those phases are handled by Grex. A user only has to define the boundary merge function (e.g., Code 5). Also, in Code 3, the `values` buffer declared in Code 1 is enabled in the reduce phase, because lazy emit is applied in our word count example.

```
int Grex::User::init(Job_t &j, Task_t& t) {
  switch ( t.phase ) {
    case MAP:
      // Do not initialize the cache for keys
      // Initialize the cache for keysize as zeros
      buffers.enable(input, keys, keysize);
      keys.cacheFunc(CACHE_NO_INIT);
      keysize.cacheFunc(CACHE_VALUE, 0);
      break;
    case GROUP:
      // Specify keys as the output buffer
      // Declare input as read-only (texture cache)
      set_targets(keys);
      input.writeEnable(false);
      break;
    case REDUCE:
      // Specify keys as the output buffer
      // Enable values
      set_targets(keys);
      buffers.enable(values);
      break;
  }
  return 1;
}
```

Code Snippet 3: User initialization

```
__device__ void Grex::User::map() {
  // Initialize active buffers
  IBUFFERS(input, keys, keysize);
  input[workerId] = tr(input[workerId]);
  // Parallel split of the input buffer into words
  KeyRecord kr  = input.split(isWordChar());
  // Emit words found in the input
  keys.emit(kr.key, kr.valid);
  // Emit key size only for boundary keys
  keysize.emit(kr.size, input.is_boundary(kr));
}
```

Code Snippet 4: User map phase function

In Code 4, the map phase is implemented as a device function executed in the GPU. (Code 4 − Code 8 are all device functions.) In Code 4, the `tr()` function is executed by the threads to convert the input data to lower-case by referring to the translation table stored in the constant memory, while removing non-word characters in parallel. In Code 4, the parallel split is invoked via the `input.split()` function provided by Grex. The pointers to the keys found as a result of executing the parallel split step is written to global memory by the `keys.emit()` function. In Code 4, only the keys are generated due to lazy emit.

```
__device__ void Grex::User::mmerge() {
  IBUFFERS(keys, keysize);
  // Read two keys on the boundary of two adjacent thread blocks
  char *k1 = keys.last_of_blk(workerId), *k2 = keys.first_of_blk(workerId+1);
  int size1 = keysize.last_of_blk(workerId);

  // If one word is spread across two adjacent blocks, merge parts
  // into one key. This is the case if k1 ends where k2 begins
  if ( k1 + size1 == k2 ) // Just delete the 2nd key to merge
    keys.del_first_key(workerId+1);
}
```

Code Snippet 5: User function for boundary merge at the end of the map phase

The boundary merge function is shown in Code 5. The function is executed in parallel using $B-1$ threads where B is the total number of the thread blocks. Note that deleting the first key of the second block is enough for a thread to merge the pieces of a key divided between the adjacent thread blocks, because the last key in the first block already points to the beginning of the merged word.

```
__device__ void Grex::User::reduce(){
  IBUFFERS(keys, values);
  // Parallel split of keys into two groups at points where two
  // consecutive keys are different
  ReduceRecord rr = keys.split(CompareKeys());
  keys.emit(rr.key, rr.valid);
  values.emit(rr.size, rr.valid);
}
```

Code Snippet 6: User reduce function

The reduce function in Code 6 splits the keys into groups and combines each group with the same key into a single <*key, value*> pair. A thread executes the user defined `compareKeys()` function to compare the key assigned to itself with the key assigned to the next thread. It returns 0 if the two keys are equal. Otherwise, it returns 1. Grex creates a bit string in shared memory, in which 1 indicates the beginning of a new group and contiguous 0's following 1 indicates a group, similar to the bit string in Figure 6. By applying the exclusive parallel prefix sum, Grex computes the size of each key group as discussed in Section 4. By doing this, the reduce function in Code 6 groups the keys and aggregates each group with the same key into a single <*key, value*> pair, e.g., <word, count> in word count. At the end of the reduce phase, actual values are generated and written to global memory.

Code 7 merges a key group divided between two adjacent thread blocks due to load balancing. If there are B thread blocks, $B-1$ threads are used for boundary merge at the end of the reduce phase. Using the user defined compare function (e.g., strcmp() in word count), a thread compares the last key of a thread block and the first key of the next block. If they are same, the frequency of the word is changed to the sum of the sizes of the first and second key groups. This parallel process is executed iteratively until no group size increases any further.

```
__device__ void Grex::User::rmerge(){
  IBUFFERS(input,values);
  char *k1=last_of_blk(workerId), *k2=first_of_blk(workerId+1);
  int  *v1=plast_of_blk(workerId), *v2=pfirst_of_blk(workerId+1);
  if (compare(k1, k2)){
    *v1+=*v2;
    return 1;
  }
  else return 0;
}
```

Code Snippet 7: User function for boundary merge after the reduce phase

```
void Grex::User::end(){
  compact(keys, values);
  char *b_data = input.download();
  char **b_keys = keys.download();
  int  *b_values  = values.download();
  display(b_keys, b_values);
}
```

Code Snippet 8: User end function

Finally, Grex invokes the `User::end()` function when all phases are executed. The example `User::end()` function given in Code 8 downloads the keys and values from the global memory in the GPU into the main memory in the host.

6. Performance Evaluation

In this section, the experimental settings for performance comparisons among Mars [7], MapCG [8], and Grex are described. Also, the performance evaluation results are discussed.

We pick Mars and MapCG for performance evaluation, because they represent the state-of-the-art in terms of GPU-based MapReduce. In addition, they are open source. Ji and Ma [11] have also developed a novel GPU-based MapReduce framework. It uses atomic operations to avoid two pass execution of a MapReduce job, similar to MapCG. However, [11] is not publicly available; therefore, we cannot compare the performance of Grex to that of [11].

We have tested the performance of Grex using different generations of GPUs for Mars and MapCG. The experiments for performance comparisons between Grex and Mars were run on the NVIDIA GeForce 8800 GTX GPU, which was used for performance evaluation by the developers of Mars [7]. The benchmarks comparing Grex against MapCG were executed on the NVIDIA GeForce GTX 460 GPU that supports atomic operations needed by MapCG. Using different platforms for Mars and MapCG provides fair comparisons, since Mars' design was influenced by relatively restrictive programming support of older generation GPUs, in which atomic operations are not available and coalescing memory accesses has a significant impact on the performance. The host machine in both cases has the AMD Athlon II X4 630 CPU, 4GB RAM, and 500GB hard disk. Linux 2.6.32.21 is installed on the host.

6.1. Benchmarks

In this section, the benchmarks used for performance evaluation are described. Table 2 summarizes the characteristics of the micro-benchmarks. They are popular MapReduce applications and used for performance evaluation in the Mars [7] and MapCG work [8]. By using the same benchmarks, we intend to provide the fairness of our performance comparisons with Mars and MapCG. Similar workloads are used for performance evaluation in [11] and [12] too.

1. Similarity Score (SS): Similarity score is a matrix application. The input is a list of document feature vectors $\vec{v_i}$ that represent the characteristics of given documents. The output of this application is pairwise similarity scores of the input documents $\frac{v_i \cdot v_j}{|v_i| \cdot |v_j|}$. Since the size of an input data is fixed in SS, no parallel split phase is needed. Similarly, the size of groups in reduce phase is known and fixed. Thus, the input split and boundary merge phases are not needed as shown in Table 2. To compute the similarity score, we implement tiled matrix operations, in which tiles (i.e., sub-matrices) are manipulated independently in the map phase and the intermediate results are aggregated in the reduce phase. The divisors and the dividends are calculated in the map phase. In the reduce phase, we calculate the final values by doing the divisions.

2. Matrix Multiplication (MM): Since the input to MM is fixed sized, it does not require the input split and boundary merge phases as summarized in Table 2. We implement tiled matrix multiplication in the map phase and add the partial sums of the tiles in the reduce stage.

3. String Match (SM): Given an input string, the string match application finds the offset of the first character for each occurrence of the string in the input file. Our implementation in Grex uses a constant buffer to store the search string. Neither sequential nor parallel input split is needed, because a match can be found without tokenizing the input data. In the map phase, parallel string matching is performed. In Grex, boundary merge is performed to find any matching string divided between the adjacent thread blocks. This application does not require sorting or reduction, since the output buffer already contains individual pointers to the matched strings.

4. Word Count (WC): This benchmark counts the frequency of each word's occurrences in the input file. The final output is <*word, frequency*> pairs sorted in non-ascending order of frequencies. This benchmark is required to run every phase of MapReduce as shown in Table 2. Grex does parallel split and boundary merge. In addition, Grex uses the lazy emit option to further decrease the memory consumption and access delay by storing only the keys until the end of the reduce phase.

Name	Input	Split	Map	B. Merge	Reduce	L. Emit	Data Size
SS	fixed	no	yes	no	yes	no	512, 1024, 2048 documents
MM	fixed	no	yes	no	yes	no	512, 1024, 2048 elements
SM	variable	no	yes	yes	no	no	32, 64, 128 MB
WC	variable	yes	yes	yes	yes	yes	32, 64, 128 MB
II	variable	yes	yes	yes	no	no	32, 64, 128 MB
PC	variable	yes	yes	yes	yes	no	32, 64, 96 MB

Table 2: Properties of the benchmarks

5. Inverted Index (II): The inverted index application reads a group of web documents and outputs every encountered link and the names of the web documents that include the link. Since the URL size is variable, this application requires the parallel split, map, boundary merge, and group phases for Grex. However, it needs no reduce phase.

6. Page View Count (PC): In the page view count benchmark, the input is a group of web logs in the form of (URL, IP address, cookie) tuples and the output is the number of distinct visits to each URL sorted in non-ascending order. The input to this benchmark is variable sized; therefore, this application requires the parallel split, map, boundary merge, and group phases. In addition, it needs the reduce phase as summarized in Table 2.

We have tested three different sets of inputs for each application: small, medium, and large, similar to Mars and MapCG. For Word Count and String Match applications, the input data is created by merging various e-books. The input for Page View Count application is created by randomly combining different web logs. The input for Inverted Index is generated using a simple crawler. The input for Matrix Multiplication and Similarity Score are randomly generated using the input generator provided by Mars [31].

6.2. Experimental Results

The performance evaluation results in terms of the speedup achieved by Grex over Mars and MapCG are given in Figures 8 and 9, respectively. We also show the throughput achieved by Grex for the large data sets in Table 2. A more detailed discussion of the performance results follows.

1. Similarity Score (SS): For this benchmark, Grex achieves up to approximately 2.4x speedup over Mars for the largest data set as shown in Figure 8. It also provides roughly 2.5x speedup over MapCG for the medium data set as shown in Figure 9. The performance loss we observe for the large data set in Figure 9 is due to the increased input size that requires the reduce phase to combine more partial results. The observed speedups are due to better use of memory hierarchy. The throughput for the largest dataset with Grex is 56 GB/s from the task release to completion.

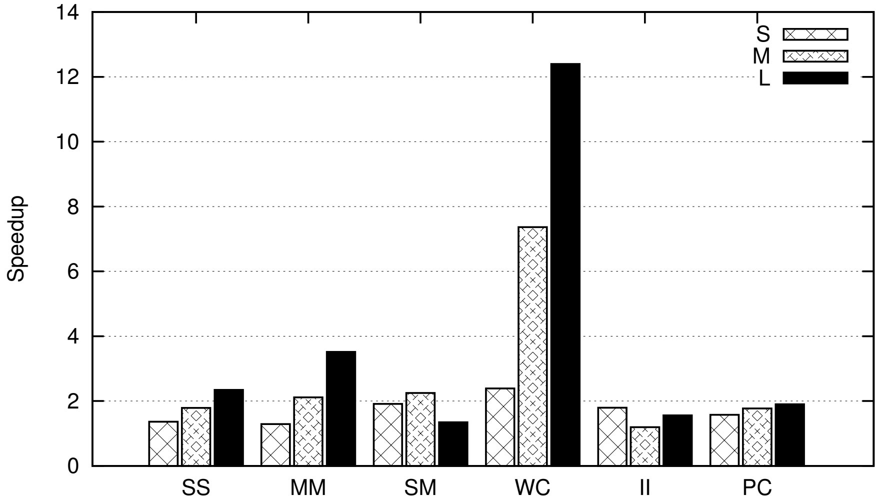


Figure 8: Speedup of Grex over Mars (Delay of Mars: 86ms - 10901ms)

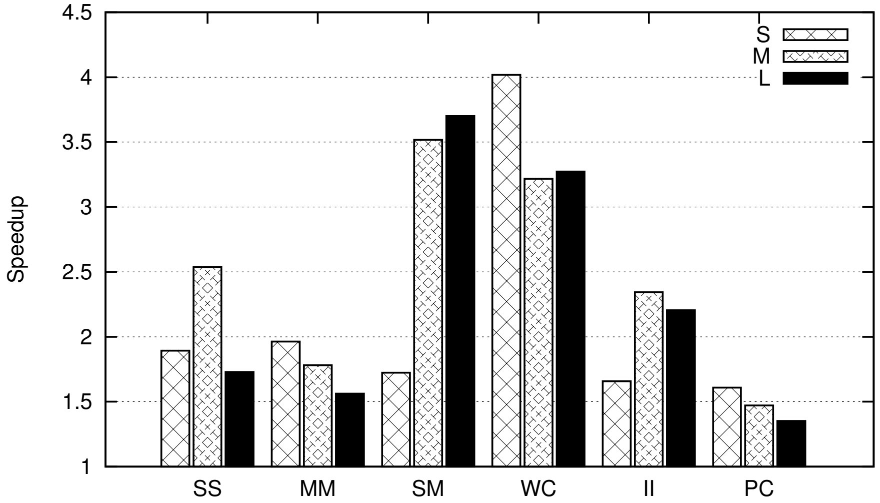


Figure 9: Speedup of Grex over MapCG (Delay of MapCG: 53ms - 2389ms)

2. Matrix Multiplication (MM): For the largest data set, which requires to multiply two 2048 x 2048 matrices, Grex is 3.8x faster than Mars. Grex shows 2x speedup over MapCG for the small set; however, the speedup decreases down to 1.5x for the large data set, similar to SS. Although SS and MM do not need parallel split or lazy emit, Grex substantially enhances the performance by taking advantage of the GPU memory hierarchy and tiling technique. For the fairness of comparisons, tiling is used for the baselines as well as Grex. The throughput of Grex is 92 GB/s for the large dataset. Both SS and MM do matrix multiplication. However, SS requires vector multiplications and divisions, which are more computationally demanding than multiplying numbers and adding the partial results in MM. Hence, for SS and MM, Grex shows different performance in terms of the throughput and speedup over Mars and MapCG.

3. String Match (SM): Grex achieves maximum 2.2x speedup over Mars for the medium-size input and 3.7x speedup over MapCG for the largest data set. Grex achieves the performance improvement due to parallel split and efficient buffering using the shared memory and texture cache. The throughput of SM for the large dataset is 36 GB/s. By comparing the throughput achieved by Grex for SS, MM, and SM, we observe that processing variable size data is more challenging than dealing with fixed size data is. As a result, Grex supports significantly lower throughput than processing fixed size data.

4. Word Count (WC): Grex achieves up to 12.4x speedup over Mars. Also, it achieves up to 4x speedup over MapCG. Among the tested benchmark applications, Grex achieves the highest performance improvement for word count. This is because word count requires all the MapReduce phases and the amount of data to process can be decreased only after the reduce phase begins. Since we use English e-books for WC, data skews impair the performance of Mars and MapCG. In contrast, Grex evenly divides intermediate <*key, value*> pairs to the reduce tasks to avoid performance penalties due to skewed data distributions. Further, Grex applies the parallel split method, while applying lazy emit as well as caching to decrease the frequency of global memory access. The throughput for WC in Grex is 14 GB/s.

5. Inverted Index (II): This benchmark requires the map and grouping phases of MapReduce to invert links and group them together; however, it needs no reduce phase. Due to parallel split, even data partitioning among map/reduce tasks, and efficient buffering, Grex is up to 1.7x and 2.3x faster than Mars and MapCG, respectively. The measured throughput of II with Grex for the large dataset is 11 GB/s. II deals with variable sized data. In addition, it has no means to reduce the amount of data during the computation, because it only inverts links without aggregating them at all. Thus, Grex achieves the lower throughput than it did for WC.

6. Page View Count (PC): For the large data set, Grex is 2x faster than Mars and 1.3x faster than MapCG. This benchmark needs all the MapReduce phases, similar to WC. However, Grex achieves relatively small performance gains over Mars and MapCG, because dealing with variable-size URLs requires more complex data processing than handling natural language words in WC. For the large dataset, the throughput of Grex is 8 GB/s. The throughput is lower than that of II, because PC is more compute-intensive than II is. In PC,

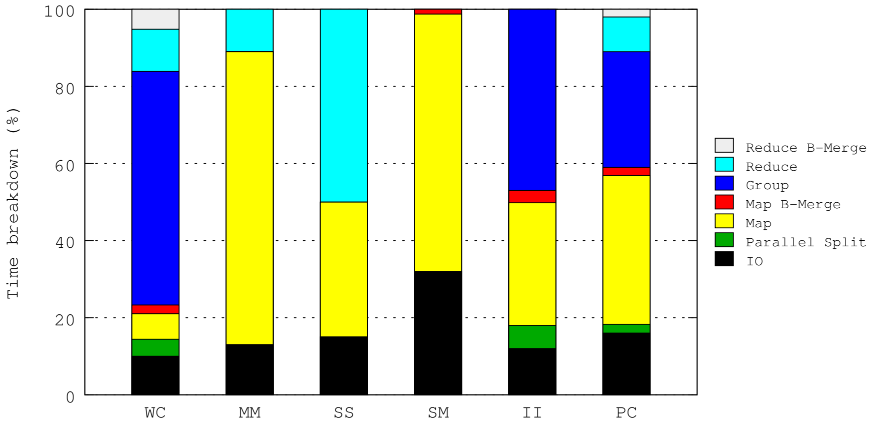


Figure 10: The time breakdown of Grex for the large data set

intermediate <*key, value*> pairs have to be sorted and aggregated in the reduce phase.

Finally, Figure 10 shows the time breakdown of the Grex phases in terms of percentage. I/O for moving input and output data between the CPU and GPU takes approximately 10% of the total job execution delay on average with the only exception of SM. In SM, however, the percentage of time spent for I/O is exaggerated due to the artifact of a computationally simple map phase. Also, SM needs neither a grouping phase nor a reduce phase. Although both MM and SS are matrix benchmarks, the relatively complex reduce phase of SS takes about 50% of the job execution time. Notably, textual data applications that need the grouping phase, i.e., word count, inverted index, and page view count, spend approximately $30\% - 60\%$ of the job execution time for sorting. However, parallel split and boundary merge phases together take less then 10% of total execution time. From these results, we claim that the overhead of parallel split and boundary merge is acceptable considering the fact that (either sequential or parallel) input split is required to process variable size data. Moreover, we observe that further performance gains can be yielded by accelerating the sort phase, while decreasing the I/O overhead. A thorough investigation is reserved for future work.

Overall, Grex has achieved substantial performance improvements compared to Mars and MapCG, because it avoids executing the map and reduce phases twice and effectively utilizes the GPU memory hierarchy, while supporting parallel split and even load distribution among map/reduce tasks. Also, Grex outperforms MapCG by avoiding serialized memory management and by supporting parallel split, even data partitioning among map/reduce workers, lazy emit, and efficient memory management.

7. Conclusion and Future Work

In this paper, we present a new GPU-based MapReduce framework called Grex. We exploit the GPU hardware parallelism and memory hierarchy to significantly enhance the performance especially in terms of the delay. We strive to avoid creating possible performance bottlenecks or load imbalance due to inefficient data distribution or data skews. At the same time, we avoid using atomics and locks, which incur potentially high contention for memory access and complexities for conflict resolution. To achieve these goals, we have developed a number of new methods for GPU-based MapReduce summarized as follows:

- Grex supports parallel split that replaces the sequential split step provided by most existing MapReduce frameworks;
- Grex partitions data to map and reduce tasks in a parallel, load-balanced fashion;
- Grex supports efficient memory access by leveraging shared memory, texture cache, and constant memory; and
- It supports lazy emit to further decrease the frequency of global memory access and the resulting delay.

We have actually implemented and experimentally evaluated Grex. The results of the performance evaluation indicate up to 12.4x and 4.1x speedup over Mars [7] and MapCG [8] that represent the current state of the art in terms of GPU-based MapReduce computing.

Given the promising initial results, we will incorporate Grex with Hadoop to efficiently process large amounts of data with minimal delays. We will also investigate the portability of Grex to AMD GPUs.

References

[1] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable Parallel Programming with CUDA, ACM Queue 6 (2) (2008) 40–53.

[2] A. Munshi, OpenCL: Parallel Computing on the GPU and CPU, Presentation at Computer Graphics and Interactive Techniques (2008).

[3] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Communications of ACM 51 (2008) 107–113.

[4] Hadoop Project, http://hadoop.apache.org.

[5] J. Lin, The Curse of Zipf and Limits to Parallelization: A Look at the Stragglers Problem in MapReduce, in: Workshop on Large-Scale Distributed Systems for Information Retrieval, 2009.

[6] Y. C. Kwon, M. Balazinska, B. Howe, J. Rolia, A Study of Skew in MapReduce Applications, in: Open Cirrus Summit, 2011.

[7] B. He, W. Fang, Q. Luo, N. K. Govindaraju, T. Wang, Mars: a MapReduce Framework on Graphics Processors, in: International Conference on Parallel Architectures and Compilation Techniques, 2008.

[8] C. Hong, D. Chen, W. Chen, W. Zheng, H. Lin, MapCG: Writing Parallel Program Portable between CPU and GPU, in: International Conference on Parallel Architectures and Compilation Techniques, 2010.

[9] C. Ranger, R. Raghuraman, A. Penmetsa, G. R. Bradski, C. Kozyrakis, Evaluating MapReduce for Multi-core and Multiprocessor Systems, in: IEEE International Symposium on High Performance Computer Architecture, 2007.

[10] R. M. Yoo, A. Romano, C. Kozyrakis., Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System, in: IEEE International Symposium on Workload Characterization, 2009.

[11] F. Ji, X. Ma, Using Shared Memory to Accelerate MapReduce on Graphics Processing Units, in: IEEE International Parallel and Distributed Processing Symposium, 2011.

[12] J. A. Stuart, J. D. Owens, Multi-GPU MapReduce on GPU Clusters, in: IEEE International Parallel and Distributed Processing Symposium, 2011.

[13] J. Talbot, R. M. Yoo, C. Kozyrakis, Phoenix++: Modular MapReduce for Shared-Memory Systems, in: International Workshop on MapReduce and its Applications, 2011.

[14] R. Chen, H. Chen, B. Zang, Tiled-MapReduce: Optimizing Resource Usages of Data-Parallel Applications on Multicore with Tiling, in: International Conference on Parallel Architectures and Compilation Techniques, 2010.

[15] S. Coleman, K. S. McKinley, Tile Size Selection Using Cache Organization and Data Layout, ACM Conference on Programming Language Design and Implementation 30 (1995) 279–290.

[16] M. de Kruijf, K. Sankaralingam, MapReduce for the Cell B.E. Architecture, Tech. rep., Department of Computer Sciences, The University of Wisconsin-Madison (2007).

[17] B. Catanzaro, N. Sundaram, K. Keutzer, A Map Reduce Framework for Programming Graphics Processors, in: In Workshop on Software Tools for MultiCore Systems, 2008.

[18] R. Farivar, A. Verma, E. M. Chan, R. Campbell, MITHRA: Multiple data Independent Tasks on a Heterogeneous Resource Architecture, in: IEEE Conference on Cluster Computing, 2009.

[19] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, J. Manferdelli, High Performance Discrete Fourier Transforms on Graphics Processors, in: International Conference for High Performance Computing, Networking, Storage and Analysis, 2008.

[20] J. Bolz, I. Farmer, E. Grinspun, P. Schröoder, Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid, ACM Transactions on Graphics 22 (2003) 917–924.

[21] R. Szerwinski, T. Güneysu, Exploiting the Power of GPUs for Asymmetric Cryptography, in: Workshop on Cryptographic Hardware and Embedded Systems, 2008.

[22] S. Manavski, G. Valle, CUDA Compatible GPU Cards as Efficient Hardware Accelerators for Smith-Waterman Sequence Alignment, BMC Bioinformatics 9 (2008) 1–9.

[23] Q. Qian, H. Che, R. Zhang, M. Xin, The Comparison of the Relative Entropy for Intrusion Detection on CPU and GPU, Australasian Conference on Information Systems (2010) 141–146.

[24] P. Bakkum, K. Skadron, Accelerating SQL Database Operations on a GPU with CUDA, in: Workshop on General-Purpose Computation on Graphics Processing Units, 2010.

[25] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, GPU Computing, IEEE Proceedings 96 (5) (2008) 879–899.

[26] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. Purcell, A Survey of General-Purpose Computation on Graphics Hardware, Computer Graphics Forum 26 (2007) 80–113.

[27] A. Bayoumi, M. Chu, Y. Hanafy, P. Harrell, G. Refai-Ahmed, Scientific and Engineering Computing Using ATI Stream Technology, Computing in Science and Engineering 11 (2009) 92–97.

[28] NVIDIA, CUDA Programming Guide 3.2, NVIDIA, 2009.

[29] CUDA Zone, http://www.nvidia.com/object/cuda_home_new.html.

[30] GPU Gems 3 - Chapter 39. Parallel Prefix Sum (Scan) with CUDA, http.developer.nvidia.com/GPUGems3/gpugems3_ch39.html.

[31] Mars: A MapReduce Framework on Graphics Processors,
http://www.cse.ust.hk/gpuqp/Mars.html.

Final Report of Louisiana Tech University Equipment Subcontract Grant

PI-s: Md Enamul Karim, Vir Phoha, Sumeet Dua

The AFOSR support Louisiana Tech University received under the grant FA 9550-09-1-0165 is utilized in developing two complementary experimentation frameworks: (i) a dual channel non-interactive protocol based lightweight framework for identification, storage and analysis of possibly malicious traffic and (ii) a generic distributed social network framework, which could be extended to other multi-sensing based network analytics. They are briefly described below:

1. A framework for identification, storage and analysis of possibly malicious(/Botnet) traffic

Investigators: Md Karim and Vir Phoha

A framework is designed based on a non-interactive dual channel lightweight protocol (***Figure 1***) we developed that can be used for the detection of botnet traffic. The realization of this protocol is achieved using Raspberry-PI attachments to keyboards (***Figure 2***) and thereby facilitating the dual channels. We collected network traffic data from systems using this scheme over three months and conducted some preliminary investigations on the collected data. The detail is available in reference [1] which will be presented at the ***20th Annual Network & Distributed System Security Symposium (NDSS 2013)***.

The protocol we have developed has wide applicably in lightweight verification and analyses of the data we have been collecting can provide some very unique understanding of the malicious traffic and possible drifts in their patterns. AFRL can benefit from our protocol and data for authentication/verification and botnet related research.

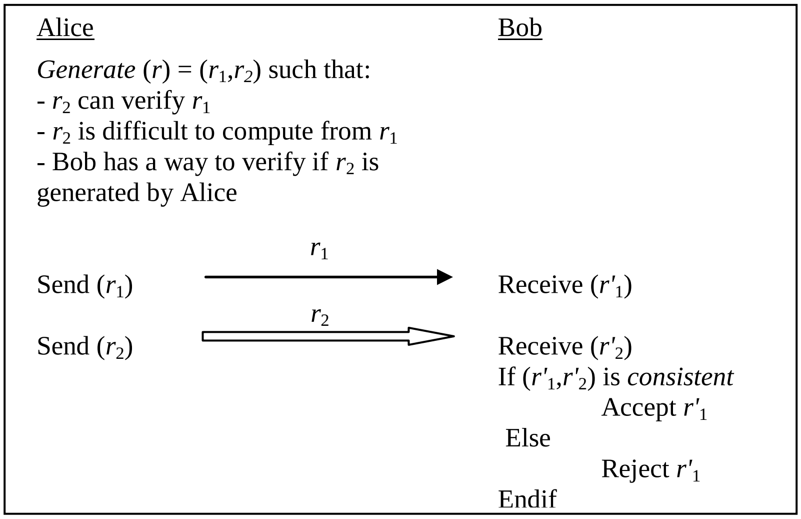
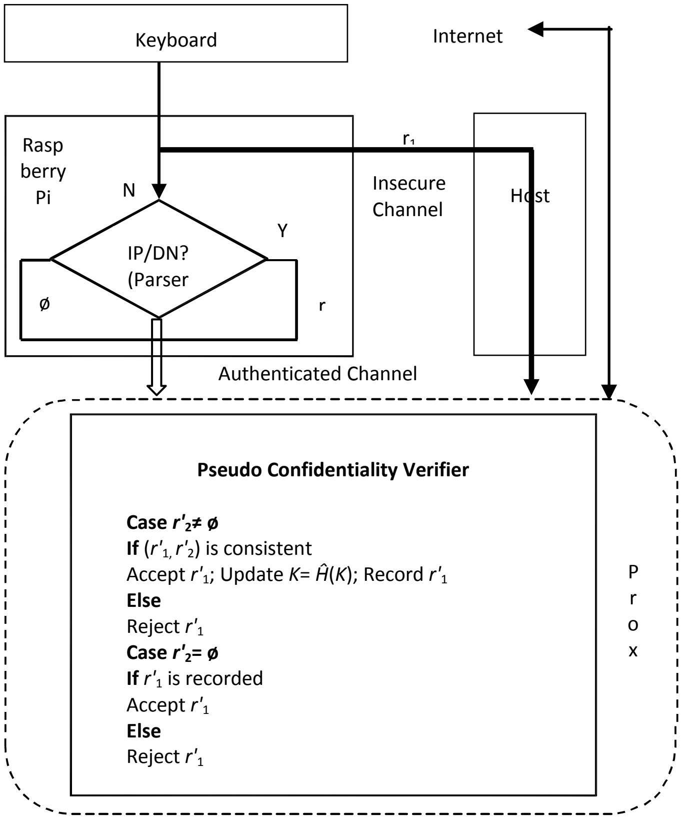

Alice
Bob
Generate (r) = (r_1,r_2) such that:
- r_2 can verify r_1
- r_2 is difficult to compute from r_1
- Bob has a way to verify if r_2 is generated by Alice
r_1
Send (r_1)
Receive (r'_1)
r_2
Send (r_2)
Receive (r'_2)
If (r'_1,r'_2) is *consistent*
Accept r'_1
Else
Reject r'_1
Endif

Figure 1. Proposed non-interactive dual channel protocol



```
Case $r'_2 \neq \emptyset$
If ($r'_1, r'_2$) is consistent
Accept $r'_1$; Update $K=\hat{H}(K)$; Record $r'_1$
Else
Reject $r'_1$
Case $r'_2 = \emptyset$
If $r'_1$ is recorded
Accept $r'_1$
Else
Reject $r'_1$
```

Figure 2. Outline of the prototype for malicious traffic detection

2. A generic distributed social network framework

Investigator: *Sumeet Dua*

A secure cyber space heavily relies on Situational Awareness (SA). SA is described as the recognition and realization of the performance of a cyber-system. An effective SA relies on the prediction of threats both internal and external to the cyber-system that does not disrupt the cyber-systems ability to deliver a set of deliverables that it is expected to support. The realization of SA on a cyber-system offers its set of challenges, of which handling the data deluge associated with such a system is vital. Data management and data analytics in a cyber-system is vital as the confidence of predicting events tied down to the functioning of the cyber-system. These predicted events are succinct to the creation of a system capable of identifying zero day attacks rather than a reactive system. ***Our efforts in situational awareness have focused on a generic distributed social network framework, which could be extended to other multi-sensing based network analytics. The equipment allowed us to accelerate our efforts in simulating and performing analytics on the social networks.***

Social networks have transcended from a means of casual communication to the glue that connects individuals over the virtual cyber space. Social networks and the associated social media provide a deluge of information that could be exploited to enhance the SA of a system. To this end, cybernetics- the science of analyzing human behavioral patterns to gauge the behavior of a community is of growing importance. Our primary has been on elaborating the theoretical underpinnings of tracking dynamic communities (groups of users) in social networks using cybernetics. Our efforts in spatio-temporal associative learning are applicable here as they provide correlational instances of user and community behavior. We have investigated techniques employed to detect new events, track events (with time), and summarize events. The following is a summary of all the activities carried out under this grant: DURIP, using the hardware acquired and tested. ***Figure 3*** provides a conceptual overview of the proposed system that supports analysis of large volumes of instructed data used in this project.

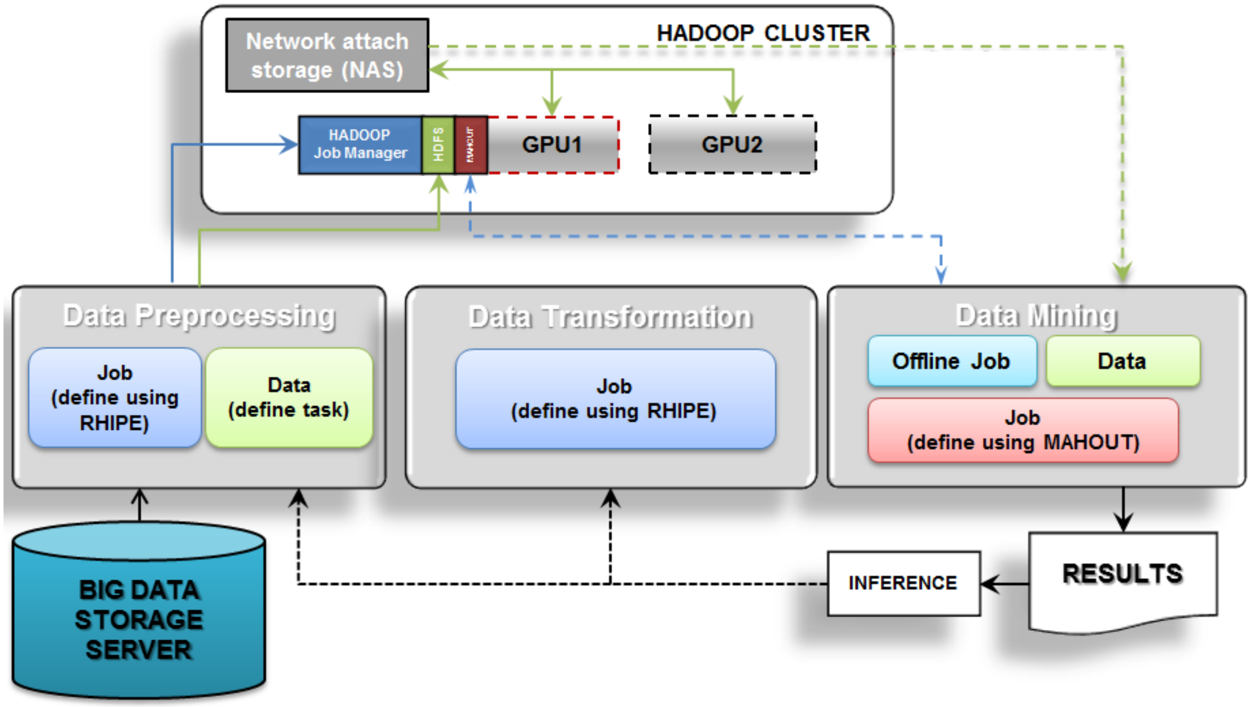


Figure 3. A schematic representation of the proposed distributed infrastructure for large scale data management

(a) <u>HADOOP CLUSTER: Set up on GPGPU's</u>

The main objective of installation of HADOOP on the two GPGPU's is to leverage the GPU's compute power in a scale out architecture. By scale out we imply that this infrastructure will provide an effective programming model for large-volume data processing. It will also provide for specialized aggregation queries using the Map-Reduce framework. This approach was chosen to enable us to manage our own network connectivity, communication, data placement and movement, load balancing, and most important fault tolerance. HADOOP with the HDFS, provides for all the above. The language R is the primary coding language used to exploit this distributed data-mining infrastructure for data preprocessing and data transformation routines. We ensure that the distributed map reduce functionality in R is provided using RHIPE and distributed data mining routines are supported using MAHOUT.

(b) <u>Big Data Storage:</u>

The Dell-Power Edge R710 storage server enables storage of raw data (large files). This nine TB storage acts as an independent data management server used to store and backup raw data.

(c) <u>Representative Projects benefiting from the equipment:</u>

The following are two representative projects that benefited from the equipment provided through the DURIP project.

i. ***Social Network Data Simulator:***

Simulation of network data is critical to the development, testing, and evaluation of observed trends, outliers, and potential threats. Developed using C++, the social network simulator is designed to generate synthetic user tagging behavior data that resembles real life social tagging behavior of users in a social network (or human interactions in an online network). Simulating individual users' concepts when reading articles generates the tags. The simulator then uses these tags to generate social links, as well as taking into account the network topography. The generic structure of the simulator will allow us to extend this to other spatio-temporal domains.

ii. ***Network Analysis:***

Higher order features in a social network are those that encompass primary features like users and articles (resources), which evolve with time. This evolution makes the selection of higher order features in an evolving community a challenge. The topics of interest are decided using the users tagging behavior of different users that change over a period of time. The challenges encountered when trying to track changes in tagging behaviors are the reduction or termination of tag usage, an addition of a newer more relevant tag to the existing resource being used, and analysis of communities with unequal number of nodes present in the network. This algorithmic framework requires data management across the storage servers and computational learning using the GPGPU platform. Future efforts will entail using time series discord detection approaches for similarity search and data filtering schemas.

3. Publication(s) resulted from this support

[1] Irakiza, D., Karim, M. E. and Phoha, V. V., A dual channel protocol for assuring pseudo-confidentiality, Network and Distributed System Security Symposium (NDSS) 2013 (accepted for short talk).

A Non-Interactive Dual Channel Authentication Protocol for Assuring Pseudo-confidentiality

David Irakiza, Md E. Karim, Vir V. Phoha
Center for Secure CyberSpace
Louisiana Tech University
{dir003,mdekarim,phoha}@latech.edu

Abstract

We introduce a non-interactive dual channel authentication protocol and apply it to long distance communication for assuring pseudo-confidentiality, a criteria that prevents a malicious agent from exfiltrating information to unauthorized destinations. Unlike previously proposed protocols that assume a manual (human-aided) or equivalent authenticated channel, our protocol utilizes a non-manual authenticated channel. We analyze the security properties for a possible realization of this protocol and develop a prototype. Through a Raspberry-Pi implementation, we show how the incorporation of the proposed scheme into the future design of keyboard interfaces may impact authentication practices.

1. Non-interactive dual channel authentication protocols

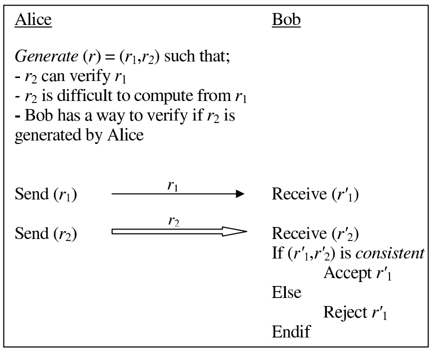
Non-interactive dual channel authentication protocols employ two channels and authenticate information r_1 received through one presumably insecure channel using a piece of brief information r_2 (computed from r_1) received through the other presumably authenticated channel. To remain lightweight these protocols employ one way communication from a message sender, Alice, to a message recipient, Bob. It can be shown that in these protocols, imposters cannot authenticate themselves if, (i) only one of the channels is compromised or (ii) both channels are compromised but attacks on them are not coordinated.

Existing literature describes a family of non-interactive message authentication protocols (e.g., [1, 2, 4, 3, 5]), known as NIMAPs, that use a manual (human-aided) authenticated channel. In these protocols, a hash value for each message being sent is generated. Alice then transmits the message and hash over the insecure and authenticated channels respectively to Bob. In some protocols, a key is applied to the hash function when generating the hash value and this key is sent with the message over the insecure channel [3, 5] or with the hash value over the authenticated channel [2].

NIMAPs assume that the authenticated channel is human-aided hence adversarial influence on this channel is significantly limited. This assumption is sufficient for short distance communications requiring infrequent authentication of messages. However, this limits their application in long distance communications (e.g, over the internet) where frequent authentication is required and a human-aided authenticated channel is unrealistic and/or infeasible.

2. Proposed protocol

Replacement of the human-aided authenticated channel in NIMAPs erodes the security benefit assured by such a channel. This exposes them (NIMAPs) to channel spoof attacks where an adversary, Eve, could spoof the identity of the authenticated channel when sending her authentication messages to Bob. Because of the non-interactive nature of



Alice

Bob

Generate (r) = (r_1,r_2) such that;
- r_2 can verify r_1
- r_2 is difficult to compute from r_1
- Bob has a way to verify if r_2 is generated by Alice

Send (r_1) $\xrightarrow{r_1}$ Receive (r'_1)

Send (r_2) $\xRightarrow{r_2}$ Receive (r'_2)

```
If ($r'_1$,$r'_2$) is consistent
        Accept $r'_1$
Else
        Reject $r'_1$
Endif
```

Figure 1. Proposed protocol (Generic)

This work is supported by AFOSR Grants FA9550-09-1-0479 and FA 9550-09-1-0165.

the protocol, Bob has no way to know whether these authentications are sent by Alice hence, he will accept a message if its authentication is valid (i.e., it authenticates the message sent through the insecure channel).

We address this issue by introducing the following: (i) we assign the task of Alice to a small hardware attachment that we assume not to be affected by a channel compromise, and (ii) we assume that it is difficult for the adversary, Eve to compute r_2 given r_1.

Figure 1 presents the proposed generic protocol. In this protocol, Alice is a hardware attachment to the keyboard with the following properties: (i) She has processing capabilities to parse typed keystrokes and apply a method *generate* to identify r_1 and to compute r_2 from r_1; (ii) She only receives input from the keyboard; (iii) She sends output to Bob using two different channels (one is insecure that runs through the host and the other is authenticated that bypasses the host). Bob is a verifier program located in a remote computer.

In Figure 1 (and in subsequent protocol figures), the “′” added to information received by Bob indicates that the sent and received values might be different and the insecure and authenticated channels are represented by “$\rightarrow$” and “$\Rightarrow$” respectively.

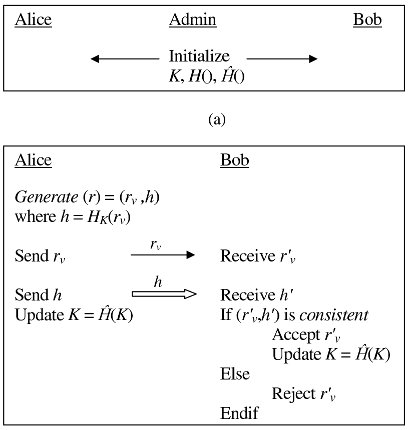
The protocol is designed based on the following assumptions: (i) an r_2 can verify the associated r_1; (ii) an adversary can generate an r_1 or snoop the r_1 generated by Alice but it is difficult for her to compute the associated r_2 from a given r_1; (iii) Bob can compute r_2 from r_1; (iv) r_1' is accepted only if (r_1', r_2') is *consistent* i.e., r_{2_b} (the expected r_2' computed by Bob for the r_1' he received) is the same as r_2'.

Alice *generates* r_1 and r_2 from the keystrokes typed by a user and sends r_1 through the possibly compromised host using the insecure channel and r_2 directly to Bob using the authenticated channel. When Bob receives r_1 and r_2 from Alice, he accepts r_1' only if (r_1', r_2') is *consistent*. To defeat this protocol Eve either (i) waits for Alice to transmit an r_1 and replaces r_1 with $r_{1_{eve}}$ and expects that it will be verified by the associated r_2 (generated and transmitted by Alice), or, (ii) estimates an $r_{2_{eve}}$ for her $r_{1_{eve}}$ and transmits both of them to Bob pretending that the $r_{2_{eve}}$ is transmitted over the authenticated channel (by spoofing the authenticated channel's identity).

Existing NIMAPs do not make assumption (ii), perhaps because the challenge involved with its realization. Instead, they assume that Eve cannot successfully use (or masquerade to be using) the manual authenticated channel. These protocols can be defeated if the manual channel is replaced by a non-manual physical channel since Eve then can compute the right $r_{2_{eve}}$ for her $r_{1_{eve}}$ and masquerade as Alice.

3. A hash based realization for assuring pseudo-confidentiality

The exfiltration of information by a malicious agent from a compromised host to an external adversary can be restricted by blocking all access from the host to unauthorized destinations. We refer to an authorized destination as a domain name/IP that: (i) has been typed at least once by a human user (establishing consent/approval), or (ii) has been returned in a response packet from an authorized destination. We define ***pseudo-confidentiality*** as the security property satisfied when an adversary cannot exfiltrate information to unauthorized destinations.



(a)

(b)

Figure 2. Hash based realization

A hash based realization of the proposed protocol assuring *pseudo-confidentiality* is shown in Figure 2. In this realization, the *generate* function used by Alice produces components r_v and h. r_v is the destination (IP/domain name) of a request being made and $h = H_K(r_v)$ is the hashed value of the requested destination. This realization works in two phases as described below:

(i) Initialization phase

This is a one step setup process required prior to the establishment of the rest of the protocol. During this phase, a human operator (i) informs Bob of the functions H and $\hat{H}$ used by Alice and (ii) initializes a K for Alice and Bob. During the protocol phase, function H uses K as a parameter to produce h and function $\hat{H}$ is used to update K by both Alice and Bob. Due to some failure or interruption in communication, if Alice and Bob do not have the same updated K, all destination requests will be rejected and the value of K will have to be re-initialized.

(ii) Protocol phase

In this phase, components r_v and h are transmitted from Alice to Bob via the insecure and authenticated channels respectively. When Bob receives both r'_v and h', he computes his own h using his K i.e., $h = H_K(r'_v)$ and considers (r'_v, h') to be *consistent* if the received $h' = h$. The value of K is updated by both Alice and Bob using another function $\hat{H}$. Alice updates her K after sending h to Bob and Bob updates his K every time he receives an $h' \neq \emptyset$ from Alice and accepts r'_v. $\hat{H}(K)$ transforms the value of K based on the current value of K to produce a new K.

During experimentation we relax the definition of *consistent* by accepting requests to destinations that are (i) non-typed but had been authorized before and (ii) received in response to requests to authorized destinations, to improve the usability of the solution.

Security Analysis

Bob accepts r'_v if (r'_v, h) is *consistent* for the current K (that Bob has knowledge of). Bob can be made to accept a request from Eve in the following cases:

Case 1: Channel spoof attack

A malicious agent, Eve, residing in the host, can no longer launch a channel spoof attack simply by sending her own r_v and h to Bob if the K she uses for computing h (at a certain instance) is not what Bob is expecting Alice to use. Since our assumption is that Eve has no knowledge of K, Alice can only expect that the K she is using matches with the K Bob is expecting. For an n-bit K, the probability of such a match is 2^{-n}.

Case 2: Second preimage Attack

Eve can replace the destination r_v generated by Alice with her own destination r_{eve} and expect that there exists a K'' such that $H(r_v, K) = H(r_{eve}, K'')$. A second preimage resistant hash function is relevant in our case because for a second preimage resistant hash function, given an input r_v, it should be difficult for an adversary to find another input $r_{eve} \neq r_v$ such that $H(r_{eve}) = H(r_v)$. If H is ϵ_s second preimage resistant (i.e., ϵ_s is the probability of $H(r_{eve}) = H(r_v)$ occurring), then the probability of a successful attack is ϵ_s.

4. Results

We setup a prototype as shown in Figure 3 using Raspberry-Pis (model B) as the hardware attachments to the keyboard and collected data for over four months. As mentioned in section 3, our experimentation considers a relaxed definition of *consistent* as reflected in the logic for *Pseudo Confidentiality Verifier* in Figure 3. Figure 4 shows typical results for a representative seven day period. As expected, all illegitimate requests were denied (requests not initiated by the user). Some legitimate requests were also denied because associated destinations were not typed once before their uses. Figure 4 (b) shows the breakdown of the legitimate typed and un-typed requests.

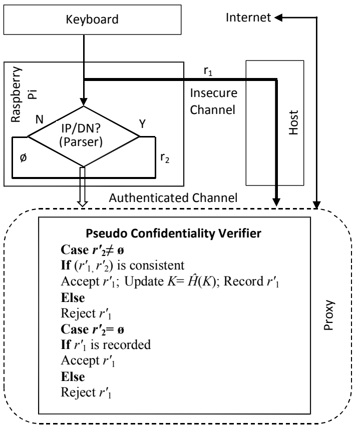
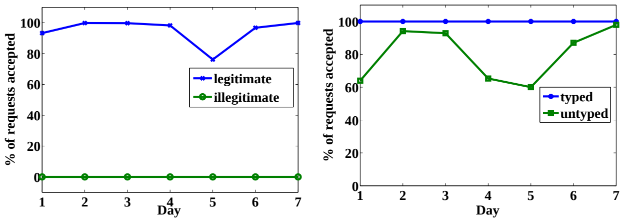


Figure 3. Raspberry-Pi based prototype



(a) (b)

Figure 4. % of requests accepted per day

References

[1] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *NDSS*, 2002.

[2] C. Gehrmann, C. J. Mitchell, and K. Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7(1):29–37, Spring 2004.

[3] A. Mashatan and D. R. Stinson. Noninteractive two-channel message authentication based on hybrid-collision resistant hash functions. *IACR Cryptology ePrint Archive*, 2006:302, 2006.

[4] S. Pasini and S. Vaudenay. An optimal non-interactive message authentication protocol. In *Proceedings of the 2006 The Cryptographers' Track at the RSA conference on Topics in Cryptology*, CT-RSA'06, pages 280–294, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] M. R. Reyhanitabar, S. Wang, and R. Safavi-Naini. Non-interactive manual channel message authentication based on etcr hash functions. In *Proceedings of the 12th Australasian conference on Information security and privacy*, ACISP'07, pages 385–399, Berlin, Heidelberg, 2007. Springer-Verlag.

Final Report of Tennessee State University Equipment Subcontract Grant

PI: Sachin Shetty

The AFOSR equipment grant was instrumental in supporting research projects in IP geolocation and cloud carrier security. The equipment will also support newly funded research projects on embedded systems security and smartphone security. In addition, undergraduate and graduate students also leveraged the equipment, to implement their respective senior design projects and theses. Following is an overview of how the equipment affected current projects, new projects, student projects and theses, future impacts and AFRL collaboration

1. Current Projects

IP Geolocation: We developed a classification based method to geolocate nodes on the internet with higher reliability, robustness and sensitivity than the current state-of-the art measurement-based IP geolocation techniques. The equipment grant allowed us to purchase two Dell PowerEdge M 520 servers (quad core) for training and testing the machine learning classifier. The presence of large number of measurements in our experiment necessitated the need for developing a parallel implementation of our classifier on quad core servers. The parallel implementation provided a faster speedup and we were able to publish results of our work in Milcom conference. Recently, we extended the work to perform visualization of IP geolocation results on Google maps. For this work, we extensively used the quad core servers to implement the complete software system. A paper based on this work has been submitted to a conference.

1) Hellen Maziku, Sachin Shetty, Keesook Han, Tamara Rogers. Enhancing the Classification Accuracy of IP Geolocation. Proc. of Military Communications Conference, Orlando, Florida, Nov 2012.
2) Asmah Muallem and Sachin Shetty. Visualization of IP Geolocation of Spam Email, submitted to IEEE ComCAP 2013

Cloud Carrier Security: In this research effort, we developed a risk assessment framework to investigate the security risk of the cloud carrier between cloud users and two commercial cloud providers. The risk assessment framework examined the security vulnerabilities of operating systems of routers within the cloud carrier. The framework provides the quantifiable security metrics of each cloud carrier which enables users to assess the security services provided by cloud providers. We used a Markov chain for our framework and the operating system vulnerability information was collected from hundreds of routers between 100 PlanetLab nodes and datacenters for two cloud providers. We used the equipment grant to purchase two quad core workstations for evaluating the Markov chain framework. We have submitted the results of this research effort to one conference.

1) Swetha Lenkala, Sachin Shetty and Kaiqi Xiong. Security Risk Assessment of Cloud Carrier, submitted to ACM CCGRID 2013

2. New Projects

The AFOSR equipment grant provided the leverage to attract two funded projects from Boeing and AFRL. The Boeing research project involves the modeling of wireless attacks on a mobile cyber physical system. An experimental testbed comprising of 10 mobile robots has been developed with normal and attack traffic. We are currently collecting RF signals which will be later used for modeling the attack behavior. The servers and workstations will be utilized to evaluate the attack model. The AFRL project focuses on the cloud based detection of malicious websites on smartphones. The nodes purchased from the equipment grant will be included in the development of a small cloud testbed to evaluate the detection mechanism.

1) "Development of an agile and resilient security architecture to protect embedded systems from emerging cyber threats", Boeing contract, 11/1/2012 - 10/31/2013.
2) "Cloud computing based detection and response system to combat malware infecting web browsers on smartphones", AFRL MLP contract. 12/1/2012 - 11/30/2013.

3. Student Projects/theses

Two undergraduate students used the equipment grant to complete their capstone senior design projects on cloud carrier security and insider threat detection. Two Master students completed their theses with the help of the resources provided by the equipment grant. Currently, 5 undergraduate and 5 graduate students are directly and indirectly benefitting from the resources provided by the equipment grant.

4. Future Impacts

The equipment grant will be very beneficial to support our current research projects and will be featured in our facilities description as we submit future proposals for funding. The equipment grant has played a crucial role in the development of the TSU cloud.

5. AFRL Collaboration

The resources from the equipment grant are currently leveraged to implement the project activities outlined in an AFRL MLP contract on smartphone security. One of the requirements of the project is the presence of a cloud environment. The equipment grant has helped us develop the cloud platform at TSU which will provide an excellent testbed to evaluate the project activities outlined in the AFRL MLP contract. We are confident that the sustainability of the equipment will be maintained by the AFRL MLP contract and similar grants/contract from AFRL and other funding agencies.

Visualizing Geolocation of Spam Email

Asmah Muallem, Sachin Shetty, and S. K. Hargrove

Abstract— **Viruses and phishing scams, as a result of spam, are increasingly becoming numerous. Spontaneous methods used by spammers present a threat in spam prevention. Tools for spam identification and prevention are increasing but lack presentation fundamentals. Online databases help determine the locations of spammers and provide more information using an IP address. The integration of components used in this process play an important role in how users use this information. A primary concern is lack of visualization tools to effectively analyze the information. In this paper, we develop a visualization tool to represent the geographical locations of spammers based on the integration of MaxMind and WhoIS databases along with the Google Maps API. The visualization tool provides a visualization of spam origination along with patterns of spammers identified from spam activity. A key component in the development of this tool is its extensible framework allowing for the addition of resources to retrieve more information about a spammer and analyze additional patterns of spammers for spam analysis.**

Index Terms— **Network Visualization, Security Visualization, Spam Analysis, IP Geolocation.**

I. Introduction

Email has been gaining popularity since the early 1990's, and has become the standard for electronic communication. Email has become the primary source of communication due to its storage, organization, and flexibility. With social networks such as Facebook and Twitter providing electronic communication, email still remains the most popular online activity. Most companies ban the use of social networks due to its flexibility of distributing both professional and non-professional material and information. Advertisers facilitate email as a means of inexpensively spreading advertisement which sometimes lead to excessive spam in a user's mailbox. Spam fills inboxes with useless emails and exposes Advertisement spam usuallyviruses and phishing scams if the advertisers webmail service or ISP is hacked. Filtering spam as a result of this problem becomes an email provider's worst nightmare. Providing an efficient spam filter helps detect new spam and remove it from the user's inbox but does not prevent it. Some problems of spam are network traffic delays, storage, and the serious security threats it poses. The techniques created to detect and prevent spam focus on analyzing patterns in datasets of email from various users for a given location. A concern for users is the privacy of emails, which requires current tools to parse and filter sensitive data from the emails prior to analyzing. The parsing of disparate and massive datasets to implement a visual analytics framework incurs additional overhead. In this paper, we deploy IP Geolocation as a tool to provide a user friendly interface for visualizing the geographical locations of spam attackers for a particular email account on a Google map with analysis capabilities and an extensible framework for additional resources to identify patterns in spam attacks.

Spam is a common type of cyber nuisance that not only wastes resources but also poses security threats [1]. Determining the geographic location of an Internet host is valuable for a number of Internet applications. [11]. IP Geolocation has found applications in cloud security, especially in the ability to accurately determine the location of data in a cloud computing environment. The use of IP Geolocation benefits network analysts, police authority, and ordinary users because of its ability to address different avenues in network security.

Recently, there have been some efforts in the area of visual analytics of emails. Analyzing and managing one's mailbox or archive have been implemented through visualizations [7, 10]. Visualization techniques designed to show patterns in incoming email which can reveal misidentified pieces of spam, common spam sources, and patterns such as periods of increased spam activity have also been implemented [2]. Analytical models that use coefficient vectors, 'density' and 'input', with visual clustering methods to classify and display the spam emails have been introduced for the analysis of spam email viruses and attacks [1]. We introduce an approach extending previous tools to identify the source of spam based not only on email information but ISP provider and region information. In this paper we present the design and implementation of a tool utilizing the techniques of IP Geolocation to visualize the geographical locations of spam attackers using a Google map with analysis controls to identify patterns of spam attackers.

Asmah Muallem is with the Electrical Engineering Department, Tennessee State University, Nashville, TN 37209 USA
(e-mail: amuallem@my.tnstate.edu).
Sachin Shetty is with the Electrical Engineering Department, Tennessee State University, Nashville, TN 37209 USA (e-mail: sshetty@tnstate.edu).
S. K. Hargrove is with the Electrical Engineering Department, Tennessee State University, Nashville, TN 37209 USA
(e-mail: skhargrove@tnstate.edu).

The organization of the paper is as follows. We present the system model in Section II. Our approach in detail is presented in Section III. In the following section we describe the implementation of our visualization system prototype. We present the results demonstrating the evaluation of the application in Section V. We conclude with future work in Section VI.

II. SYSTEM MODEL

Our model is based on visualizing the geographical location of spam attackers using a web application. A primary element in preventing spam is investigating the source. Our system model (Fig. 1) comprises of data acquisition, database design, and visualization phases. We process spam information from email accounts through the web application and generate a visualization of the geographical locations of the spam attackers.

The first phase is data acquisition and is part of the spam generation process. We generate spam email using a test email account and gather the information when the user logs into the email account through the web application. We use a MySQL database to store the emails. We integrate into our database MaxMind and WhoIS information (geolocation, ISP, etc) corresponding to the spam emails gathered. The last and most important phase is visualization. Finally, in the visualization phase, relevant data associated with the email account is retrieved to display the geographical locations of the spam attackers along with analysis controls on a Google map. The visualization analysis controls help determine patterns in spam attacks (for ex., frequency of attacks from a given location or ISP over a period of time) or provide additional information to perform further analysis and help prevent future spam attacks.

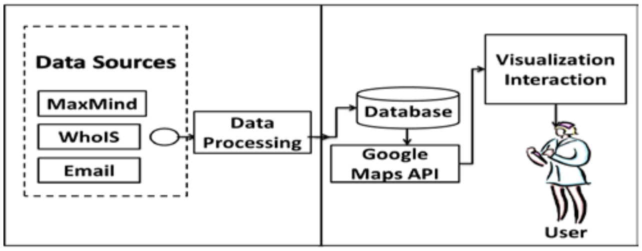


Fig. 1. System model

III. APPROACH

Our objective is to design a visualization tool to determine the geographical locations of spam attackers for analysis purposes and spam prevention. Our system model presents a novel approach for preventing spam by integrating databases and visualization tools to come up with a prototype for visualizing the location of spam attackers. Through this model we can detect spam attackers which use different email addresses associated with the same domain to send spam emails.

Our approach provides detailed information about the spam attacker along with the geographical location to identify the main source of spam and give further analysis to help determine a pattern. The current web tools use IP addresses to query databases for information about spam attackers but do not have the capability of obtaining all information at one time or from one place. This process involves using multiple websites to retrieve the IP address of the spam attacker and the information about the spam attacker. This method can cause discrepancies since different databases and websites are used in the process. Our approach enhances this capability by offering one tool to implement all of these actions along with providing a user friendly visualization and analysis controls to gather more information about the spam attacker.

A. Data Acquisition

We create a test account in Gmail to initiate the process of retrieving spam. We believe Gmail will serve as the best email provider for our research since it is associated with several online services such as YouTube. YouTube, blogs, and online discussion groups can lead to excessive spam emails in a user's mailbox. Gmail's spam filter is decent but does not always automatically mark spam emails as spam and these emails can still be delivered to the user's mailbox if they have different email address but the same domain name. We select outlook as the second email provider due to its flexibility in allowing several different email accounts. Gmail's spam folder is equivalent to Outlooks' Junk E-mail folder. We create Hotmail and Gmail email accounts to sign up with online services such as YouTube and join several online and Google discussion groups. We generate spam a few months prior to the implementation of the tool. During the first month, Gmail's test account accumulated approximately 200 or more spam email and Hotmail's test account accumulated approximately 100 or more junk mail.

B. Database Design

We use MySQL to store and manipulate data. MySQL is an open source relational database management system and can be integrated with programming languages in Visual Studio and Java. We store MaxMind data prior to storing spam email information. MaxMind's IP Geolocation data and services provide Geo IP which consists of country, region, postal code, city, area code, and latitude/longitude coordinate information for a given IP address. Geo IP data are composed of user-centered location data gathered through online sites that store web visitor's geographic locations. Millions of datasets run through algorithms to identify, extract, and produce location points for IP addresses.

MaxMind GeoIP offers paid and free services and some of the free services include downloadable files of Geo IP data in csv and binary formats and APIs for retrieving and manipulating Geo IP data. Our database imports downloaded GeoLite City Geo IP data files. GeoLite City contains two csv files: blocks, and location. The location file includes country, city, state, region, postal code, area code, and latitude/longitude information. The blocks file includes IP addresses in long format. A column called location is included in both files to match the IP address range to a location. Our database also integrates WhoIS information. WhoIS queries databases that store registered users or assignees of an Internet source. The query is initiated using telnet and a successful acknowledgement returns the ISP and domain registered information for a given IP address.

We use ASP.Net and C# along with ADO.Net to connect to MySQL to import downloaded MaxMind GeoIP data and store spam email information into the MySQL database. We use Koolwired, an open source DLL of a C# implementation of IMAP, to connect to Gmail's IMAP configuration through C# and the Microsoft.Office.Interop.Outlook library, which provides support for interoperability between the COM object model of Microsoft 2010 and managed applications that automate Outlook, to connect to Outlook. We parse and gather spam from the user's Junk E-mail/spam folder through the C# application using the user's login credentials after importing MaxMind Geo IP data. We store sender, receiver, subject, and date/time information. We also store additional information such as transport headers from Outlook emails. Transport headers are useful for further analysis to identify a pattern in the encoding of spam emails. We determine the geographical location coordinates of the spam attacker during the process of storing spam email information into our database.

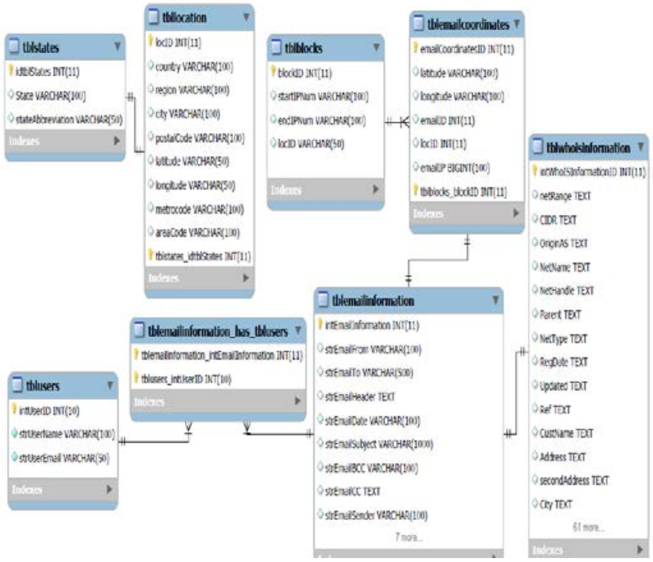


Fig. 2. MySQL database design

We use a MySQL Routine, insertEmailCoordinates, to identify the location coordinates associated with the IP address range in MaxMind for the spam attacker's IP address. We use the C# method, Dns.GetHostAddresses (), to translate the parsed email domain name into an IP address. We convert the IP address to long format to allow for the capability of passing the IP address to insertEmailCoordinates for retrieving location coordinates. We obtain WhoIS information about the spam attacker using the translated IP address in the last step of the database design phase. We gather information from WhoIS using a telnet session initiated through the C# web application and query the WhoIS database with the translated IP address as a parameter. The WhoIS information returns in different formats based on the IP address, in which we address by designing an algorithm to handle all formats and prevent errors. The tables within our MySQL database include tbllocation, tblblocks, tblusers, tblwhoisinformation, tblemailinformation, and tblemailcoordinates which are illustrated in Fig. 2.

C. *Visualization*

Our tool can help detect patterns of spam attackers through our analysis and visualization controls. We implement the visualization of geographical locations for spam attackers using the C# web application responsible for the process discussed in the database design phase. We provide visualization by manipulating collected data and using the Google Maps API. The main source for the visualization is based on the geographical location coordinates determined from insertEmailCoordinates and WhoIS information stored in our database during the database design phase. We connect to the database using ADO.Net, retrieve the WhoIS information and geographical coordinates and store them in a C# data structure. The JavaScript functions in ASP.net parse the information from the data structure and send it to the Google Maps API functions. We design JavaScript functions in the ASP.Net code to generate multiple markers on the Google map identifying locations of different spam attackers with info windows for each marker providing detailed information about each spam attacker.

Currently, we are only displaying WhoIS information related to the IP address in the info window. We are in the process of providing more detailed information about the spam attacker, and other map control options for different views of the map. A single marker is displayed on the Google map to represent multiple spam attackers from the same location, preventing the duplication and excessiveness of markers on the Google map. Our research uses latitude and longitude coordinates to display the locations on the Google map versus using regions. Currently, our tool provides filtering of geographical locations of spam attackers based on domain name, ISP, and the region associated with the most spam. We are in the process of providing more tools in the filtering panel for temporal analysis and filtering spam attacker locations by date, country, city, state, and email or IP address. We use different color coding schemes to represent regions with the most spam emails. We will extend the map options for different visualization features to include the map options available in the Google Maps API. We can help identify patterns of spam attackers through the analysis controls within this tool. We illustrate the visualization components in Fig. 3.

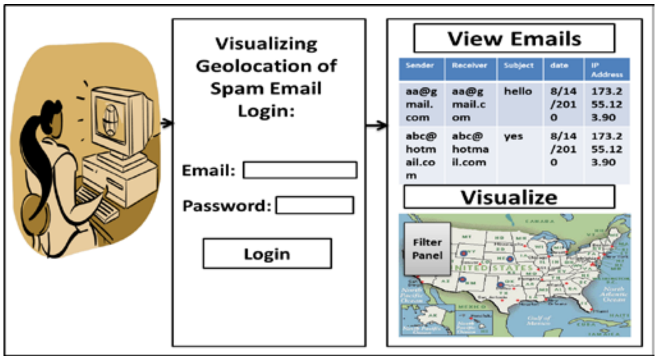


Fig. 3. Visualization components of tool.

IV. IMPLEMENTATION

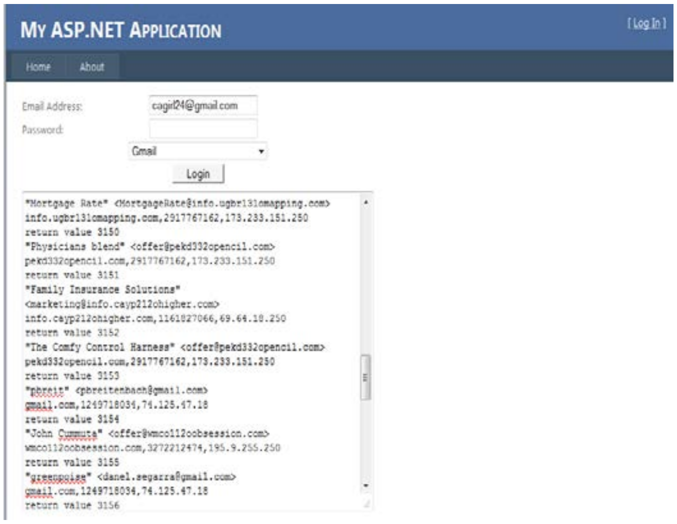
A prototype of our implementation uses test accounts created in Gmail and Hotmail. We import MaxMind Geo IP data into the database using the process discussed in the database design phase. We select the Gmail test account during the initial phase of implementation since it contains more spam emails. We log into the test email accounts through a login interface within the C# web application to initiate the process discussed in the database design phase for storing spam email and WhoIS information, and identifying geographical location coordinates. During implementation, we found discrepancies between MaxMind and WhoIS for a small percentage of IP addresses when geographical location coordinates were returned from MaxMind. We implemented further analysis on these discrepancies using the MySQL Workbench GUI and found false positives because the location of the IP address returned from MaxMind did not match the location registered with WhoIS. We resolve this issue by identifying multiple addresses in WhoIS for the IP address to give one address which will match the location returned from MaxMind. We also found locations in MaxMind which did not have an address in WhoIS.

Our analysis determined that this was due to lack of fresh updates from MaxMind since latest updates are only available monthly. We resolve this issue by comparing the registered date for the IP address from WhoIS against the date of when the latest MaxMind files were imported, the location which has the most current date is selected. Our next step in implementation is to visualize the geographical location of spam attackers from the spam email information gathered after logging into the Gmail account from the C# web application. The C# web application provides an option to visualize geographical locations for spam attackers associated with the given email account and to view stored spam email information. We are currently using an ASP.Net text box to view stored spam email information for test purposes, but will add an ASP.Net Gridview for a more organized view.

V. RESULTS

We visualize geographical locations of spam attackers after viewing the list of collected spam email information. The visualization contains multiple markers on the Google map representing the geographical locations of spam attackers along with info windows providing WhoIS information invoked when clicking the marker. The Visualization produced 3 markers on the Google map to represent the 200 spam emails in the Gmail test account. The markers represented were based off of spam attackers with different IP addresses but from the same geographical location preventing duplicate markers on the map. We filter by the Gmail domain returning 2 markers representing geographical locations of spam attackers using Gmail accounts. We also filter by the klhh333oextinction.com domain, returning 1 marker representing a spam attacker located in Arizona. The ISP filter is used to filter spam attacker locations based on the registered ISP in WhoIS. The last step in our evaluation involves checking the option to highlight the region with the most spam email, which is CA.

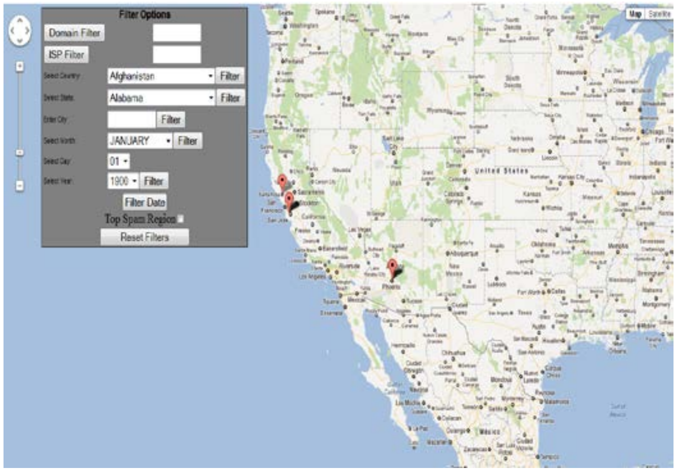
Logging into the Gmail test email account, storing email information into the database, and displaying associated email information is displayed in Fig. 4. Visualizing the Geolocation of Spam attackers for the test email account on the Google map is displayed in Fig. 5. Zooming into a desired geographical location on the Google map is shown in Fig. 6. Filtering by the Gmail.com domain to visualize spam attackers using Gmail.com and viewing WhoIS information related to the spam attacker is shown in Fig. 7. Filtering by the klhh333oextinction.com domain to visualize spam attackers using klhh333oextinction.com and viewing WhoIS information related to the spam attacker is shown in Fig. 8. The last figure displays checking the top ISP region checkbox to highlight the region with the most spam.



Email Address: cagirl24@gmail.com
Password:
Gmail

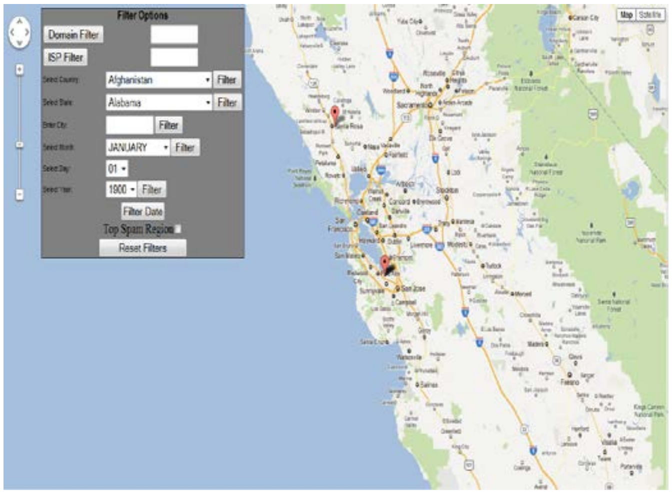
```
"Mortgage Rate" <MortgageRate@info.ugbr131omapping.com>
info.ugbr131omapping.com,2917767162,173.233.151.250
return value 3150
"Physicians blend" <offer@pekd332opencil.com>
pekd332opencil.com,2917767162,173.233.151.250
return value 3151
"Family Insurance Solutions"
<marketing@info.cayp212ohigher.com>
info.cayp212ohigher.com,1161827066,69.64.18.250
return value 3152
"The Comfy Control Harness" <offer@pekd332opencil.com>
pekd332opencil.com,2917767162,173.233.151.250
return value 3153
"pbreit" <pbreitenbach@gmail.com>
gmail.com,1249718034,74.125.47.18
return value 3154
"John Cummuta" <offer@wmco112oobsession.com>
wmco112oobsession.com,3272212474,195.9.255.250
return value 3155
"greenpoise" <danel.segarra@gmail.com>
gmail.com,1249718034,74.125.47.18
return value 3156
```

Fig. 4. Logging into the Gmail test account, processing spam email information and displaying stored information.



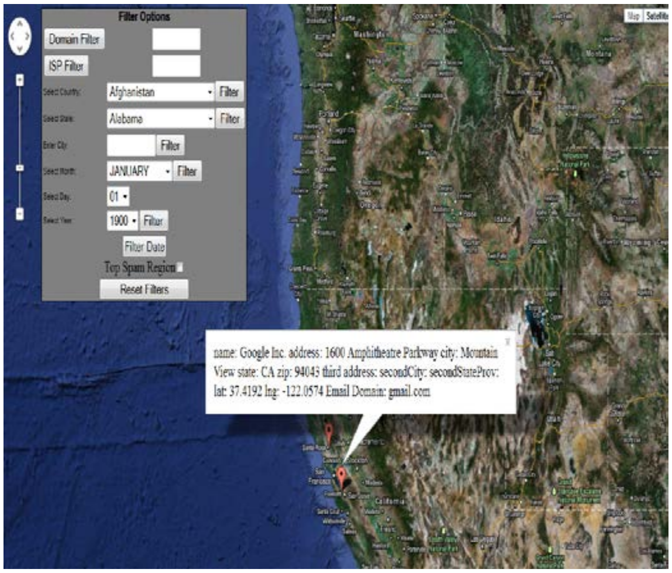
Select Country: Afghanistan
Select State: Alabama
Select Month: JANUARY
Select Day: 01
Select Year: 1900

Fig. 5. Visualizing geolocation of spam email.



Select Country: Afghanistan
Select State: Alabama
Select Month: JANUARY
Select Day: 01
Select Year: 1900

Fig. 6. Zooming into a desired location on the map.

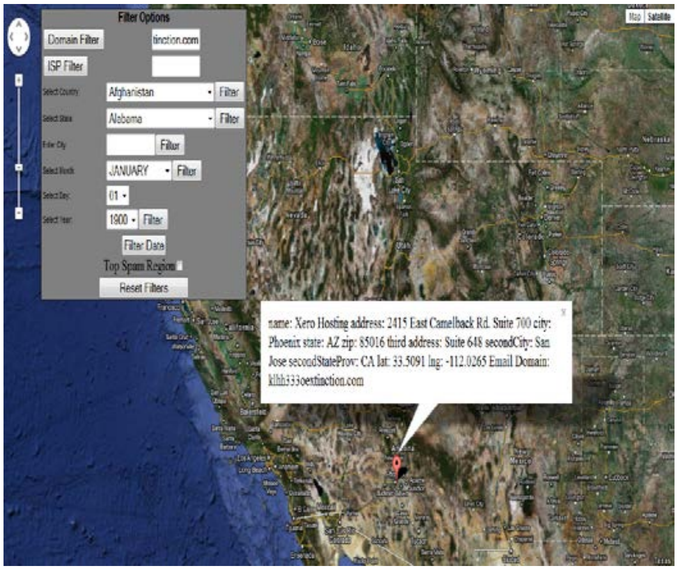


Select Country: Afghanistan
Select State: Alabama
Select Month: JANUARY
Select Day: 01
Select Year: 1900

name: Google Inc. address: 1600 Amphitheatre Parkway city: Mountain View state: CA zip: 94043 third address: secondCity: secondStateProv: lat: 37.4192 lng: -122.0574 Email Domain: gmail.com

Fig. 7. Filtering by the Gmail.com domain and visualizing geolocation of spam email associated with a Gmail account

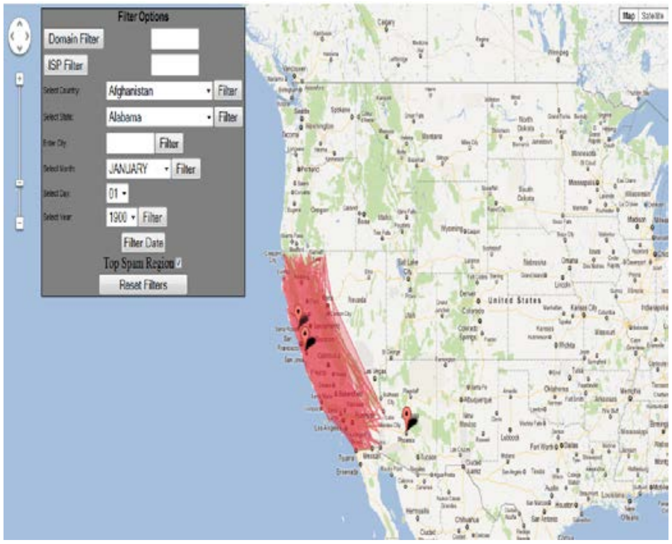
along with viewing the WhoIS information related to the account.



Domain Filter: tinction.com
Select Country: Afghanistan
Select State: Alabama
Select Month: JANUARY
Select Day: 01
Select Year: 1900

name: Xero Hosting address: 2415 East Camelback Rd. Suite 700 city: Phoenix state: AZ zip: 85016 third address: Suite 648 secondCity: San Jose secondStateProv: CA lat: 33.5091 lng: -112.0265 Email Domain: klhh333oextinction.com

Fig. 8. Filtering by the klhh333oextinction.com domain and visualizing geolocation of spam email associated with a klhh333oextinction account while using the satellite map option.



Select Country: Afghanistan
Select State: Alabama
Select Month: JANUARY
Select Day: 01
Select Year: 1900

Fig. 9. Visualizing geolocation of spam email for the Gmail test account with the region with the most spam highlighted.

VI. Conclusion

In this paper we present a visualization environment to visualize the geographical locations of spam attackers on a Google map using IP Geolocation, the integration of MaxMind and WhoIS databases, and the Google Maps API. Our results validate the effectiveness of integrating multiple systems to develop an application for preventing spam. Our analysis controls within this visualization tool can help network analysts and other users identify patterns of spam attackers. Our future work focuses on developing a visualization environment to visualize geographical locations of spam attackers based on analysis of spam emails in a cloud computing environment.

Acknowledgment

The research of Sachin Shetty and S. Keith Hargrove is supported by NSF HBCU-UP Targeted Infusion HRD #1036307 and AFOSR grant FA9550-09-1-0165.

References

[1] M. L. Huang, Q. V. Nguyen, J. Zhang, J. Wang, "Visual Clustering of Spam Emails for DDoS Analysis," *in Proc. of IEEE INFOVIS,* July 2011.

[2] C. Muelder, K. Ma, "Visualization of Sanitized Email Logs for Spam Analysis," *in Proc. of IEEE APVIS,* Feb 2007.

[3] X. Li, W. Yan, Y. Xue, "Sentinel: Securing Database from Logic Flaws in Web Applications," *in Proc. of ACM CODASPY,* Feb. 2012.

[4] X. Li, Y. Xue, "BLOCK: A Black-box Approach for Detection of State Violation Attacks Towards Web Applications," *in Proc. of ACM ACSAC,* Dec. 2011.

[5] S. Laki *et al.*, "Spotter: A Model Based Active Geolocation Service," *in Proc. IEEE INFOCOM,* April 2011.

[6] M. Samiei, J. Dill, A. Kirkpatrick, "EzMail: Using Information Visualization Techniques to Help Manage Email," *in Proc. IEEE IV,* July 2004.

[7] B. Kerr, "THREAD ARCS: An Email Thread Visualization," *in IEEE INFOVIS,* Oct. 2003.

[8] S. Rohall *et al.*, "ReMail: A Reinvented Email Prototype," *in ACM CHI,* April 2004.

[9] Z. Kan, C. Hu, Z Wang, G. Wang, X. Huang, "NetVis: A Network Security Management Visualization Tool Based On Treemap," *in Proc. IEEE ICACC,* March 2010.

[10] S. L. Rohall *et al.,* "ReMail: A Reinvented Email Prototype," *in Proc. ACM CHI,* April 2004.

[11] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, C. Huang, "Towards Street-Level Client-Independent IP Geolocation," *in USENIX NSDI, vol. 5, no. 5.,* March 2011.

[12] K. S. Ross, C. A. McCabe, R. E. Roth, "A near real-time visualization for understanding spatio-temporal patterns of violent crime in the District of Columbia," http://www.geovista.psu.edu/

Final Report of UIUC Equipment Subcontract Grant

PI: Negar Kiyavash

The proposed instrumentation for UIUC provided the PI with computational capabilities to analyze large datasets pertaining to various research tasks. Specifically the following grants of PI benefitted directly from the equipment:

- AFOSR, Information-Theoretic Causal Coordination (2010-2013)
- AFOSR, MURI: Multi-Layer and Multi-Resolution Networks of Interacting Agents in Adversarial Environments (2010-2015)
- AFOSR, YIP: Information-theoretic Approaches to Network Forensics (2011-2013)

The aforementioned proposals address information-theoretic approaches to information fusion using point process models and fundamentals from information theory and Bayesian estimation as well as performing inference tasks in networks using the timing information. One such inference task is discovering patterns of causal interactions among say a group of cooperating agents or performing network information forensics such as botnet detection and intrusion detection.

While PIs contributions are mostly theoretical, validating the theory requires extensive simulations with large time series. For instance to characterize and investigate mitigation techniques for timing side channels that arise when a resource is shared among users in packet networks, PI and her colleagues implemented a remote traffic analysis attacks which was implemented using the DURIP provided testbed. Moreover, to perform causal inference in networks, PI has proposed provably-good information-theoretic metrics for measuring statistical causal relationship devised efficient algorithms for identifying the network structure when some information about the topology is available. The proposed equipment provided PI with a strong computational platform that enabled the algorithmic development and testing of the proposed framework and algorithms.

Some of the resulting research has been submitted for publication and appears in the following manuscripts:

- C. Quinn, T. Coleman, and N. Kiyavash, ``Efficient Methods to Compute Optimal Tree Approximations of Directed Information Graphs," submitted.
- S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam, ``Mitigating Timing based Information Leakage in Shared Schedulers," submitted.
- N. Kiyavash, F. Koushanfar, T. Coleman, and M. Rodrigues, ``A Timing Channel Spyware for the CSMA/CA Protocol," to appear in IEEE Transactions on Information Forensics and Security.

Delay Optimal Policies Offer Very Little Privacy

Sachin Kadloor[†*] and Negar Kiyavash[‡*] [†] ECE Department and Coordinated Science Lab.
[‡] ISE Department and Coordinated Science Lab.
[*]University of Illinois at Urbana-Champaign
{kadloor1,kiyavash}@illinois.edu

***Abstract*—Traditionally, scheduling policies have been optimized to perform well on metrics such as throughput, delay and fairness. In the context of shared event schedulers, where a common processor is shared among multiple users, one also has to consider the *privacy* offered by the scheduling policy. The privacy offered by a scheduling policy measures how much information about the usage pattern of one user of the system can be learnt by another as a consequence of sharing the scheduler. In [1], we introduced an estimation error based metric to quantify this privacy. We showed that the most commonly deployed scheduling policy, the first-come-first-served (FCFS) offers very little privacy to its users. We also proposed a parametric non-work-conserving policy which traded off delay for improved privacy. In this work, we ask the question, *is a trade-off between delay and privacy fundamental to the design to scheduling policies? In particular, is there a work-conserving, possibly randomized, scheduling policy that scores high on the privacy metric?* Answering the first question, we show that there does exist a fundamental limit on the privacy performance of a work-conserving scheduling policy. We quantify this limit. Furthermore, answering the second question, we demonstrate that the round-robin scheduling policy (a deterministic policy) is privacy optimal within the class of work-conserving policies.**

I. Introduction

In multi-tasking systems where a finite resource is to be shared, a scheduler dictates how the resource is divided among competing processes. Examples of systems which have schedulers include, a computer where the CPU needs to be shared between the different threads running, a cloud computing infrastructure with shared computing resources, a network router serving packets from different streams etc.. Some of the commonly used schedulers are first-come-first-served (FCFS), round-robin (RR), shortest-job-first (SJF) and priority schedulers. Performance of a scheduler is measured in one of several metrics including, throughput (number of job completions per unit time), average delay (the difference between the job completion time and the job arrival time), fairness (a metric to measure if the resource is being distributed equally/fairly between the processes), etc.. A scheduler often has to make a calculated trade-off among these conflicting metrics.

We consider the scenario when a scheduler is serving jobs from two users, where one of them is an innocuous user and other a malicious one. The malicious user, Bob, wishes to learn the pattern of jobs sent by the innocuous user, Alice. Bob exploits the fact that when the processor is busy serving jobs from Alice, his own jobs experience a delay. As shown in Figure 1, Bob computes the delays experienced by his jobs and uses these delays to infer about the times when Alice tried to access the processor, and possibly the sizes of jobs scheduled. Learning this traffic pattern from Alice can aid Bob in carrying out traffic analysis attacks. The scheduling system thus incidentally creates a *timing based side channel* that can be exploited by a malicious user. In [2], the authors consider the scenario where a client is connected to a rogue website using a TOR network. The website modulates the traffic sent to the client. The side channel considered here exists in the intermediate routers. They show that an eavesdropper can exploit it to figure out the identity of the client talking to the website, thus defeating the purpose of TOR. While that attack is no longer viable [3], the reason is that there are many more TOR nodes now than they were when [2] was published, and not because the timing based side channel has been eliminated. In [4], the authors exploit the side channel in a DSL router to infer the website being visited by the victim. A similar side channel exists within Amazon's EC2 cloud computing service, which is exploited in [5]. Other works on traffic analysis include: recovery of information about keystrokes typed [6], words spoken over VoIP [7], and utilizing the timing variations required for cryptographic operations to recover cryptographic keys [8]. Motivated by these attacks, we argue that while choosing a scheduler, one has to consider the privacy offered by it along with the other performance based metrics. This should especially be the case when the scheduler serves processes from several non-trusting users, e.g., a scheduler used in a cloud computing infrastructure.

In this paper, we study a generic shared scheduler shown in Figure 1. For such systems, in order to minimize the information leakage, one has to design 'privacy preserving' scheduling policies. As a result of high correlations between the arrivals of one user and the waiting times of the other, FCFS is an example of a bad policy in this respect, [1]. An example of a good privacy preserving scheduling policy is the time division multiple access (TDMA), where a user is assigned a fixed service time regardless of whether he has any jobs that need to be processed or not. As expected, the waiting times of jobs issued by one are independent of the others' arrivals, and consequently, the policy leaks no information. However, TDMA is a highly inefficient policy in terms of throughput and delay, especially when the traffic is varying. It is especially inefficient when the number of users using the scheduler is large [1]. FCFS and TDMA represent two extremes of the trade-off between the information leakage and

This work was funded in part by grants FA 9550-11-1-0016, FA 9550-10-1-0573, 727 AF Sub TX 0200-07UI, and FA 9550- 10-1-0345.

efficiency (in terms of delay or throughput).

Scheduling policies in which the server never idles as long as there is an unserved job in the system are said to be *work-conserving* or *non-idling*. Examples of work-conserving policies include FCFS and round-robin (RR). On the other hand, scheduling policies in which the server is allowed to stay idle even when there are unserved jobs in the system are said to be *non-work-conserving* or *idling* policies. Both the TDMA and the accumulate and serve policies (derived in [1]), those that offer a guaranteed privacy, are non-work-conserving. *Is delay an inevitable price that needs to be paid for guaranteed privacy? Instead, could the scheduler use a private source of randomness to obfuscate the attacker?*

When all the jobs are of the same size, it can be shown that all work-conserving non-preemptive[1] policies incur the same average delay [9]. Also, policies that idle incur a delay which is strictly greater than that incurred by a work-conserving policy. In addition, the delay offered by an idling policy is strictly larger than the delay offered by a work-conserving policy. Work-conserving policies therefore represent a class of throughput and delay-optimal scheduling policies. In this paper, we address the question: *How does the most secure work-conserving scheduling policy stack up against TDMA on the privacy metric?*

A. Outline of the paper

In Section II, we formally introduce a system model, and the metric of performance that we use to compare the privacy of different scheduling policies. In Section II-A, we quantify the highest degree of privacy that any scheduling policy can guarantee (work-conserving, or otherwise), and demonstrate that TDMA provides the highest privacy. The privacy performance of TDMA is used to benchmark all other scheduling policies. Next, we turn our attention to the class of work-conserving policies. Consider a fictitious policy that knows the identity of the attacker, and gives priority to jobs issued by him. In Theorem 3.1, we demonstrate that such a policy is a privacy optimal scheduling policy within this class. We show that any attack that can be carried out against this policy can suitably be modified and carried out against any other work-conserving scheduling policy as well, incurring the same error for the attacker. This fact is used to bound the privacy offered by any work-conserving scheduling policy. In Section III-A, we discuss an attack against this policy. The resulting error incurred, denoted by $\mathcal{E}^{c,\text{upper},\epsilon}_{\text{Priority}}$, serves as an upper bound on the privacy performance of all work-conserving policies, and in particular, the privacy offered by round-robin, denoted by $\mathcal{E}^{c,\lambda}_{\text{RR}}$. In Section IV, we consider the privacy performance of the round-robin policy and construct a lower bound to it, denoted by $\mathcal{E}^{c,\text{lower}}_{\text{RR}}$. Numerically computing $\mathcal{E}^{c,\text{upper},\epsilon}_{\text{Priority}}$ and $\mathcal{E}^{c,\text{lower}}_{\text{RR}}$ is not straightforward. In Section V, we relate the computation of these errors to a combinatorial counting problem which can then be solved numerically. It is then argued that a parameter of the attack, ϵ can be chosen suitably so that the upper bound on $\mathcal{E}^{c,\lambda}_{\text{RR}}$ matches the lower bound exactly, thus proving the optimality of the round-robin policy on the privacy metric. However, as shown in Figure 5, there is a large gap between the privacy offered by round-robin and that by TDMA. Therefore, if the delay offered by a scheduling policy is of a higher importance than the privacy offered by it, i.e., if one is looking for a policy within the class of work-conserving policies, then the round-robin policy is a good candidate. Otherwise, if the privacy offered by it is of a higher importance, one has to pay the price of increased delay. Finally, in Section V-A, we discuss the implications of our work.

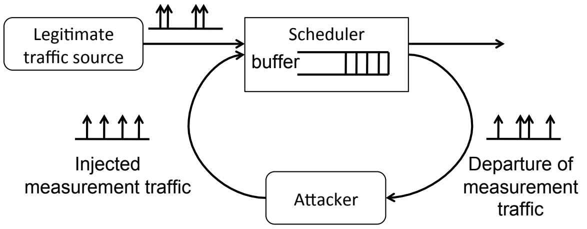

Legitimate traffic source
Scheduler
buffer
Injected measurement traffic
Attacker
Departure of measurement traffic

Fig. 1. An event/packet scheduler being exploited by a malicious user to infer the arrival pattern from the other.

This work builds on our earlier works [1], [9]. In [1], we develop a formal framework to study the information leakage in shared schedulers. In that work, it was shown that the FCFS (a work-conserving policy) scheduling policy does leak significant information, while TDMA (an idling policy) leaks the least. We also proposed and analyzed a provably secure scheduling policy, called accumulate-and-serve, another idling policy, which traded off delay for improved privacy.

In this work, we ask the question, *is there a work-conserving scheduling policy that fares high on the privacy metric?* This is the same question we ask in [9]. The major difference between that work and the current one (and also between [1] and the current one) is the metric used to quantify the privacy offered by a scheduling policy, which is discussed in the following section. The results derived in Section III of this paper are similar to those derived in [9]. However, the proofs are different owing to the difference in the definition of privacy metric. Also, the results provided in this paper subsume those presented in [9], and are much stronger.

II. System Model and Definitions

The system model is a scheduler shared by two users, the innocuous user, Alice, and the malicious one, Bob. The scheduler can serve jobs at a rate of one per unit time. Alice issues unit sized jobs to the scheduler according to a Poisson process of rate λ_2. The total number of jobs issued by Alice until time u is given by $\mathcal{A}_{\text{A}}(u)$. The malicious user, Bob, also referred to as the attacker, issues his jobs at times $t_1^n \doteq \{t_1, t_2, \ldots, t_n\}$, and is free to choose their sizes, $s_1^n \doteq \{s_1, s_2, \ldots, s_n\}$, as well. Let $t_1^{'n} \doteq \{t_1^{'}, t_2^{'}, \ldots, t_n^{'}\}$ be the departure times of these jobs. Bob makes use of the observations available to him, the set $\{t_1^n, s_1^n, t_1^{'n}\}$ and the knowledge of the scheduling policy used, in estimating Alice's arrival pattern. The arrival pattern of Alice is the sequence

[1] Non-preemptive policies are those in which the processing of a job is never interrupted once it starts getting served. Throughout this paper, we will only consider non-preemptive policies.

$\{X_k\}_{k=1,2,\ldots,N}$, where $X_k = \mathcal{A}_{\mathrm{A}}(kc) - \mathcal{A}_{\mathrm{A}}((k-1)c)$, is the number of jobs issued by Alice in the interval $((k-1)c, kc]$, referred to as the k^{th} clock period of duration c. Nc is the time horizon over which the attacker is interested in learning Alice's arrival pattern.

The privacy offered by a scheduling policy is measured by the long run estimation error incurred by Bob in such a scenario when he is free to decide the number of jobs he issues, times when he issues them and their sizes, subject to a rate constraint, and when he optimally estimates Alice's arrival pattern. Formally, the privacy offered by a scheduling policy is defined to be:

$$\mathcal{E}^{c,\lambda}_{\text{Scheduling policy}} = \lim_{N\to\infty} \min_{n,t_1^n,s_1^n:\frac{\sum_{i=1}^{n} s_i}{Nc}<\lambda} \frac{1}{N}\sum_{k=1}^{N} \mathbf{E}\left[\left(X_k - \mathbf{E}\left[X_k|t_1^n, t_1^{'n}, s_1^n\right]\right)^2\right], \quad (1)$$

where, the expectation is taken over the joint distribution of the arrival times of Alice's jobs, the arrival times and sizes of jobs from the attacker and his departure times. This joint distribution is in turn dependent on the scheduling policy used, which is known to the attacker. Finally, λ is the rate restriction on arrivals from Bob. Furthermore, the attacker is assumed to know the statistical description of Alice's arrival process. A scheduling policy is said to preserve the privacy of its users if the resulting estimation error for the attacker is high.

In this work we only consider the case when there are two users of the system, the innocuous user and the attacker. From a privacy perspective, the two user scenario is the worst case. It is true that if there are more users of the system, the attacker can only learn about the cumulative arrival pattern from all the users. However, as the authors in [5] state, in such systems, the attacker typically waits for a time when he can be assured that the victim is the only other user of the scheduling system and launches an attack then. A policy that fares well on the privacy metric in the two user scenario is also guaranteed to perform well in the multiple user scenario.

Note that we allow the attacker to choose the size of the jobs that he issues. This is in contrast to our previous works, [1] and [9], where he could only issue jobs of size one. The results derived in this work are therefore stronger in the sense that a policy that is secure on this metric is also provably secure on the earlier metric.

A. The maximum estimation error that the attacker can incur

With the metric of privacy as defined in (1), it is easy to quantify the maximum estimation error any rational attacker would incur, as show in Theorem 4.1 in [1]. By ignoring all the observations available to him, *viz.*, $\{t_1^n, s_1^n, t_1^{'n}\}$, and estimating X_k using its statistical mean alone, $\lambda_2 c$, the attacker incurs an error equal to the variance of X_k, which is also equal to $\mathcal{E}^c_{\text{Max}} \doteq \lambda_2 c$. Hence, $\mathcal{E}^c_{\text{Max}}$ serves as a benchmark against which other scheduling policies can be compared.

Also, as shown in Section IV in [1], the time-division-multiple-access (TDMA) scheduling policy achieves this bound. This is because, when TDMA scheduling policy is used, the departures of one user are completely independent of the arrivals from the other. Therefore, TDMA is a privacy optimal scheduling policy. However, as discussed in Section III of [9], because TDMA is non-work-conserving, it loses out on performance based metrics such as throughput region and delay. In the two user scenario, the rate at which each user issues jobs needs to be less than 0.5 in order for the system to be stable. A system is said to be stable if the number of unserved jobs in the system does not blow to infinity. However, if the scheduler instead used a work-conserving policy, the system would be stable as long as the sum of the rates at which the two users issued their jobs is less than 1. Also, unless the arrivals are periodic, TDMA incurs large delays.

In the subsequent section, we identify the most secure work-conserving scheduling policy and characterize its privacy metric.

III. The most secure work-conserving policy

In this section, we derive a bound on the privacy performance of any work-conserving policy. We do so by showing that if the scheduler were allowed to pick any work-conserving policy that served jobs from both Alice and the attacker, the best strategy for it is to pick the policy that gives priority to jobs from the attacker. Therefore, analyzing the performance of the priority policy serves as a bound on the performance of any other work-conserving scheduling policy. Although this policy is not implementable, because the scheduler would not know the identity of the attacker, analyzing the privacy performance of this fictitious policy gives a bound on the privacy performance of all work-conserving policies.

Theorem 3.1: A scheduling policy that gives priority to jobs from the attacker is a privacy optimal scheduling policy within the class of non-idling policies. That is, if $\mathcal{WC}$ is the class of all work-conserving policies, and $\mathcal{E}^{c,\lambda}_{\text{Priority}}$ is the privacy metric of the policy that gives priority to jobs from attacker,

$$\mathcal{E}^{c,\lambda}_{\mathrm{P}} \leq \mathcal{E}^{c,\lambda}_{\text{Priority}}, \quad \forall \mathrm{P} \in \mathcal{WC}. \quad (2)$$

Proof: A proof is given in Appendix A ■

In the following section, we specify one attack against the priority policy and compute the resulting error.

A. An attack against the priority policy

Without loss of generality, we will assume that at time 0 the system is completely empty, i.e., there are no outstanding jobs from any of the users. At time 0, the attacker issues a job. From then on, he injects a new job one time unit after the completion of his previous job. Formally, let $t_1, t_2, \ldots$ be the times when attacker injects jobs into the system, and $t_1^{'}, t_2^{'}, \ldots$ be their departure times. Then, $t_1 = 0$, and,

$$t_{k+1} = t_k^{'} + 1, \quad k = 1, 2, \ldots \quad (3)$$

The size of all jobs is ϵ, a parameter which will be specified later. At the rate the attacker issues his jobs, the system can be shown to be stable when Alice's arrival rate is less than $1-\epsilon$. Therefore, ϵ has to be chosen to be less than $1-\lambda_2$ so that the system is kept stable.

Analysis of the estimation error incurred: Note that, for some job k, if $t_k^{'} - t_k = \epsilon$, i.e., if the k^{th} job goes into service immediately after it is issued, then it must be the case that the system was empty when the job was issued, at time t_k. A busy period of the scheduling system is an interval when the processor is busy serving jobs of wither of the users. The following lemma states that, through this attack, Bob learns the start and end times of all the busy periods. Define $r_1^L \doteq \{r_1, r_2, \ldots, r_L\}$ to be the start times of the busy periods until time Nc, and let $r_1^{'L} \doteq \{r_1^{'}, r_2^{'}, \ldots, r_L^{'}\}$ be the end times of these periods.

Lemma 3.2: The start and end times of the busy periods can be computed by the attacker. Formally, r_1^L and $r_1^{'L}$ are a deterministic function of the arrival and departure times, t_1^n, and $t_1^{'n}$.

Lemma 3.3: Given the end times of the busy periods, the arrival and departure times of the attacker's jobs can be computed. Formally, t_1^n and $t_1^{'n}$ are a deterministic function of $r_1^{'L}$.

Proof: The proofs of these two lemmas are given in appendices B and C respectively. ∎

As a consequence of Lemmas 3.2 and 3.3, we have $\mathbf{E}[X_k | t_1^n, t_1^{'n}] = \mathbf{E}[X_k | r_1^{'L}]$. Therefore, the estimation error incurred by the attacker is the estimation error incurred in estimating the arrival pattern knowing the start and end times of the busy periods. In addition to the jobs issued by Alice, the jobs issued by the attacker also determine the duration of the busy periods. It can be shown that the attacker is better off estimating the arrival pattern of Alice if the busy periods are of short duration. When the busy periods are long, there are many possible arrival patterns that lead to a busy period of the same duration, leading to higher estimation error for the attacker. Notice that the results of Lemmas 3.2 and 3.3 hold true for all values of ϵ, the size of jobs issued by the attacker. Consequently, as shown in Section V, in order to estimate Alice's arrival pattern with highest precision, the attacker should choose the value of ϵ, the size of his jobs, to be as small as possible.

Denote by $\mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon}$ the resulting estimation error incurred by the attacker. We will defer the computation of the best estimate, $\mathbf{E}[X_k | r_1^{'L}]$, and the resulting estimation error, $\mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon}$, to Section V. Since $\mathcal{E}_{\text{Priority}}^{c,\lambda}$ is the smallest error the attacker can incur among all the attacks that he can possibly launch, and $\mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon}$ is the error incurred by launching one specific attack, we have $\mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon} \geq \mathcal{E}_{\text{Priority}}^{c,\lambda}$, $\forall \lambda \leq 1-\epsilon$. Also, as a consequence of (2), $\mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon} \geq \mathcal{E}_{\text{P}}^{c,\lambda}, \quad \forall \text{P} \in \mathcal{WC}, \forall \lambda \leq 1-\epsilon$. In particular, $\mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon}$ bounds the privacy performance of the round-robin policy. In the following section, we will provide a lower bound on the estimation error incurred by any attacker against the round-robin scheduling policy, denoted by $\mathcal{E}_{\text{RR}}^{c,\text{lower}}$. Therefore, the following set of inequalities follows:

$$\mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon} \geq \mathcal{E}_{\text{Priority}}^{c,\lambda} \geq \mathcal{E}_{\text{RR}}^{c,\lambda} \geq \mathcal{E}_{\text{RR}}^{c,\text{lower}}, \forall \lambda \leq 1-\epsilon. \tag{4}$$

In Section V, it will be shown that $\lim_{\epsilon \to 0} \mathcal{E}_{\text{Priority}}^{c,\text{upper},\epsilon} = \mathcal{E}_{\text{RR}}^{c,\text{lower}}$, thus proving that the bound computed on $\mathcal{E}_{\text{Priority}}^{c,\lambda}$ in this section is tight. And more importantly, that round-robin is a privacy optimal scheduling policy within the class of work-conserving policies.

IV. Privacy performance of the round-robin policy

The round-robin scheduling policy serves jobs from multiple users as follows. Suppose there are m users issuing jobs to the scheduler, indexed 1 through m. After completion of a job issued by user i, the scheduler works on a job from user $i+1$, if present. If there are no jobs from user $i+1$, the scheduler works on a job from user $i+2$, and so on. This is known to be a 'fair' policy [10], and because it is non-idling, it is also throughput optimal. In this section, we will show that it is also optimal on the privacy metric within the class of work-conserving policies. We start by constructing a lower bound to $\mathcal{E}_{\text{RR}}^{c,\lambda}$, the estimation error incurred by the strongest attacker against the round-robin policy. We do so by providing the attacker with a side information. Without this extra information, the attacker can only be worse off in his estimation.

A. A lower bound on the $\mathcal{E}_{\text{RR}}^{c,\lambda}$

Consider a round-robin scheduler where Alice is the only user of the system. The times when she issues her jobs is given by the cumulative arrival process $\mathcal{A}_{\text{A}}(u)$, where u indexes time. Let $\mathcal{D}_{\text{A}}^1(u)$ denote the total amount of service received by Alice until time u in this system. Let $\mathcal{A}_{\text{A}}$ and $\mathcal{D}_{\text{A}}^1$ represent the functions $\mathcal{A}_{\text{A}}(u), \forall u$ and $\mathcal{D}_{\text{A}}^1(u), \forall u$ respectively. $\mathcal{A}_{\text{A}}$ is a counting function, and $\mathcal{D}_{\text{A}}^1$ is a non-decreasing function with a slope either 0 or 1 (the function is differentiable almost everywhere). If the slope is 1 at a time u, the processor is busy serving a job then. If the slope is 0, then the processor has finished serving all the jobs issued by Alice till then (scheduler never idles). Note that $\mathcal{D}_{\text{A}}^1 \leq \mathcal{A}_{\text{A}}$ and consequently, $\mathcal{D}_{\text{A}}^1(u)$ is a lower bound on the total number of jobs that have arrived from Alice till time u. Now, consider a round-robin scheduler that is used both by Alice and Bob, as shown in Figure 2. Suppose Alice's arrivals are the same as in the earlier system. Let t_1^n be the times when Bob issues his jobs, s_1^n be their sizes, and $t_1^{'n}$ be their departure times. Denote by $\mathcal{D}_{\text{A}}^2(u)$ the total service received by Alice until time u for this system. Having chosen his arrival times and sizes, and having observed their departure times, Bob has to estimate the total number of arrivals from Alice in a clock period k. We will consider the scenario where the attacker is given $\mathcal{D}_{\text{A}}^1$ as a side information as shown in Figure 2. We will show that the resulting estimation error for this attacker is a lower bound on $\mathcal{E}_{\text{RR}}^{c,\lambda}$.

Theorem 4.1: The estimate $\mathbf{E}[X_k | t_1^n, s_1^n, t_1^{'n}]$ is an inferior estimate compared to $\mathbf{E}[X_k | \mathcal{D}_{\text{A}}^1]$. Therefore, if the attacker is given a side information, the function $\mathcal{D}_{\text{A}}^1$, his own arrival and departure times give him no further information about Alice's arrival pattern, and consequently can be discarded.

We first prove the following two lemmas which form the basis of the proof of the theorem stated above.

Lemma 4.2: The departure times of jobs issued by the attacker, $t_1^{'n}$, are a function of their arrival times, t_1^n, their sizes s_1^n, and $\mathcal{D}_{\text{A}}^1$.

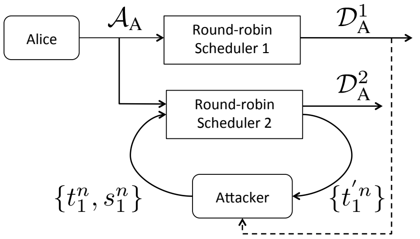


Fig. 2. Pictorial representation of the computation of $\mathcal{E}^{\mathrm{c,lower}}_{\mathrm{RR}}$, a lower bound on $\mathcal{E}^{c,\lambda}_{\mathrm{RR}}$. Apart from the information available to him through his attack, the attacker is also given a side information (shown in dotted arrow) which are the departures if Alice was the only user of the scheduling system.

Lemma 4.3: The arrival times t^n_2 are a function of t_1, s_1, and $\mathcal{D}^1_{\mathrm{A}}$. Also, t_1 and s_1 are independent of $\mathcal{D}^1_{\mathrm{A}}$ and X_k.

Proof: The proofs of these lemmas are presented in Appendix D-A and D-B, respectively. ■

Proof of Theorem 4.1: To prove the theorem, first note that the estimate $\mathbf{E}[X_k|t^n_1, s^n_1, t'^n_1, \mathcal{D}^1_{\mathrm{A}}]$ is a superior estimate of X_k compared to $\mathbf{E}[X_k|t^n_1, s^n_1, t'^n_1]$ (more information the attacker has, more accurate his estimation of Alice's traffic pattern will be). As a result of Lemmas 4.2 and 4.3, $\mathbf{E}[X_k|t^n_1, s^n_1, t'^n_1, \mathcal{D}^1_{\mathrm{A}}] = \mathbf{E}[X_k|t_1, s_1, \mathcal{D}^1_{\mathrm{A}}] = \mathbf{E}[X_k|\mathcal{D}^1_{\mathrm{A}}]$. ■

As a consequence of Theorem 4.1, we have,

$$\begin{aligned}\forall\{t^n_1, s^n_1\}, \quad & \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k|t^n_1, s^n_1, t'^n_1]\right)^2\right] \\ & \geq \ \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k|\mathcal{D}^1_{\mathrm{A}}]\right)^2\right], \qquad (5)\end{aligned}$$

and therefore, $\mathcal{E}^{c,\lambda}_{\mathrm{RR}} \geq \mathcal{E}^{\mathrm{c,lower}}_{\mathrm{RR}}$, where,

$$\mathcal{E}^{\mathrm{c,lower}}_{\mathrm{RR}} \doteq \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k|\mathcal{D}^1_{\mathrm{A}}]\right)^2\right].$$

In the following section, we present a technique to compute $\mathcal{E}^{c,\mathrm{lower}}_{\mathrm{RR}}$ numerically.

V. Computation of the Best Estimate and the Resulting Estimation Error

In this section, we present an algorithm to compute $\mathbf{E}[X_k|\mathcal{D}^1_{\mathrm{A}}]$, and $\mathcal{E}^{c,\mathrm{lower}}_{\mathrm{RR}}$ numerically. We do so by specifying an equivalence between the estimation problem, and an equivalent combinatorial path counting problem. The same technique can also be used to compute $\mathcal{E}^{\mathrm{c,upper},\epsilon}_{\mathrm{Priority}}$.

Recall that $\mathcal{E}^{c,\mathrm{lower}}_{\mathrm{RR}}$ is the estimation error incurred by an attacker who is given the departure process of a scheduler used only by Alice. Note that this additional side information is equivalent to providing the attacker with the start and end times of the busy periods of a scheduler used only by Alice. We start by considering a slightly different problem of constructing the best estimate of the number of arrivals within a busy period. Formally, suppose a busy period of duration $B+1$ is initiated at time 0, where B is some non-negative integer. Suppose an attacker observes that the processor is busy from time 0 to time $B+1$, and he knows that there is only one user issuing jobs to the system. He wishes to estimate the number of arrivals between times (u_1, u_2) where, $0 \leq u_1 < u_2 \leq B+1$. The job that initiates the busy period arrives at time $u_0 = 0$. Let $u_1, u_2, \ldots, u_B, u_{B+1}$ be the arrival times of the next $B+1$ jobs. Then, in order to sustain the busy period of duration $B+1$, the arrival times of these jobs should satisfy $u_1 < 1, u_2 < 2, \ldots, u_B < B$ and $u_{B+1} > B+1$. This is because, the job from Alice that arrived at time zero goes into service immediately and departs at time one. Therefore, in order for the busy period to sustain beyond time one, there must be atleast one more arrival by then. By a similar argument, the second job has to arrive before time two, and so on. Finally, for the busy period to end, we need $u_{B+1} > B+1$. Figure 3 shows two possible arrival patterns (shown as counting functions) that could lead to a busy period of duration 5. In fact, any counting function that lies in the green region (lightly shaded region) leads to a busy period of duration 5. In the figure, the arrival at time 0 is not shown.

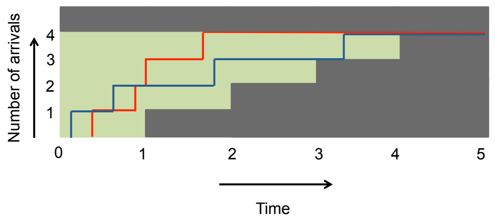


Fig. 3. Two counting functions that lead to a busy period of duration 5.

Let $\{N_s\}_{s\geq 0}$ be a Poisson process of rate λ_2. For a positive integer t, and non-negative integers $i, j, j \geq i$, define

$$\delta_{i,j}(t) = \Pr(N_t = j, N_s \geq s, s = 1, 2, \ldots, t-1 | N_0 = i).$$

$\delta_{i,j}(t)$ is the probability that a Poisson counting function of rate λ_2 jumps to state j by time t given that it starts at state i at time 0 while staying above the boundary $N_s = s, s = 1, 2, \ldots, t-1$. It can be shown that the expected number of arrivals by time $\tilde{t}$ within a busy period of duration $B+1$ is given by $\sum_{i=\tilde{t}}^{B+1} i\delta_{0,i}(\tilde{t})\delta_{i-\tilde{t},B-\tilde{t}}(B-\tilde{t})$. Computation of $\delta_{i,j}(t)$ is necessary in order to compute the best estimates and the resulting error. However, deriving a closed form expression for $\delta_{i,j}(t)$ is not easy other than some special cases. We transform the problem of computing $\delta_{i,j}(t)$ to a combinatoric path counting problem which admits a numerical solution. Before doing so, we state the following theorem which gives an equivalence between a Poisson counting process, and a Geometric approximation to it which is used later.

Lemma 5.1: Let $\{Y^n_i\}_{i=0,1,2,\ldots}$ be a sequence of i.i.d. Geometric random variables indexed by integer n, with $\Pr(Y^n_i = k) = p^k(1-p), k = 0, 1, 2, \ldots$. Define $\tilde{N}^n_s = \sum_{i=0}^{\lfloor sn \rfloor} Y^n_i$. Let p scale with n such that $np = \lambda$, where λ is a constant. As defined earlier, let N_s be a Poisson process of rate λ_2. Then, for any finite integer k, and any $0 < t_1 < t_2 < \ldots < t_k$, the joint distribution of $\tilde{N}^n_{t_1}, \tilde{N}^n_{t_2}, \ldots, \tilde{N}^n_{t_k}$ converges to the joint distribution of $N_{t_1}, N_{t_2}, \ldots, N_{t_k}$ as $n \to \infty$.

Proof: Refer Section 2.2.5 and in particular, Theorem 2.2.4 and Corollary 2.2.1 of [11]. The reference gives a Bernoulli approximation of a Poisson process. The proof for the Geometric random variable is very similar. ■

Lemma 5.1 states the following. Suppose a particle moves on a lattice, moving right a distance of $\frac{1}{n}$ and moving up a distance of k with probability $(\frac{\lambda_2}{n})^k(1-\frac{\lambda_2}{n})$, $k = 0, 1, 2, \ldots$. Then, the path of such a particle follows a Poisson process of rate λ_2 in the limit $n \to \infty$. In Figure 4, in green, we plot the sample path of a particle that moves in this fashion.

The equivalence between computation of $\delta_{i,j}(t)$ and a combinatoric counting problem is given in the following:

$$
\begin{aligned}
\delta_{i,j}(t) &= \Pr(N_t = j|N_0 = i) \\
&\quad\times\frac{\Pr(N_t = j, N_s \geq s, s = 1, \ldots, t-1|N_0 = i)}{\Pr(N_t = j|N_0 = i)} \\
&= e^{-\lambda_2 t}\frac{(\lambda_2 t)^{j-i}}{(j-i)!} \\
&\quad\times \lim_{n\to\infty}\frac{\Pr(\tilde{N}^n_t = j, \tilde{N}^n_s \geq s, s = 1, \ldots, t-1|\tilde{N}^n_0 = i)}{\Pr(\tilde{N}^n_t = j|\tilde{N}^n_0 = i)} \qquad (6) \\
&= e^{-\lambda_2 t}\frac{(\lambda_2 t)^{j-i}}{(j-i)!} \\
&\quad\times \lim_{n\to\infty}\frac{\#\text{Paths}(0,i) \to (nt,j)\text{ avoiding the boundary}}{\#\text{Paths}(0,i) \to (nt,j)}. \qquad (7)
\end{aligned}
$$

(6) follows from the results of Lemma 5.1. The denominator of the fraction on the right side of the equality in (6) is the probability that a particle starting at the point $(0, i)$ and moving according to the Geometric process described before hits the point (tn, j). The numerator is the probability that a particle starting at $(0, i)$ hits the point (tn, j) while staying above the boundary $\{(a, b) : b = \lceil \frac{a}{n} \rceil - 1\}$. Note that for this geometric process, there are a finite number of paths between the two points. In all these paths that originate at $(0, i)$ and terminate at (tn, j), the particle 'moves right and up' $j - i$ times and 'moves right without jumping' $nt - (j - i)$ times. Therefore, the probability of the particle taking any of these paths is the same, equal to $(\frac{\lambda_2}{n})^{j-i}(1 - \frac{\lambda_2}{n})^{nt}$. Therefore the ratio of the two probabilities is just equal to the ratio of the number of lattice paths on a grid that avoid a boundary to the total number of lattice paths between the aforementioned points. (7) therefore follows, where the boundary is the set of integer points (a, b) which satisfy $\{b = \lceil \frac{a}{n} \rceil - 1\}$. In Figure 4, the red line corresponds to the boundary.

The significance of Lemma 5.1 is that, for any finite n, the number of lattice paths can be counted. It is easy to see that the denominator in (7) is given by $\binom{nt+j-i}{j-i} = \frac{t^{j-i}}{(j-i)!}n^{j-i} + o(n^{j-i})$, where $\lim_{n\to\infty} o(n^{j-i})/n^{j-i} = 0$. There is no closed form expression for the numerator in (7) though. Counting the number of lattice paths between two points on a grid while avoiding a boundary is an extensively studied combinatorial problem with several applications, refer [12]. Using the lemma stated below, we can show that the numerator in (7) is given by $\gamma(i, j, t)n^{j-i} + o(n^{j-i})$, and the value of $\gamma(i, j, t)$ can be computed exactly.

Lemma 5.2 (Lemma 3A of Chapter 1 in [12]): The number of paths dominated by the path $\mathbf{p}$ with vector $(a_1, a_2, \ldots, a_n)$ can be recursively calculated as V_n using the recursion formula

$$V_k = \sum_{j=1}^{k}(-1)^{j-1}\binom{a_{k-j+1}+1}{j}V_{k-j}, \qquad V_0 = 1. \qquad (8)$$

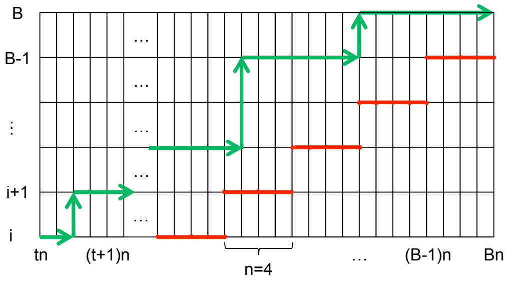


Fig. 4. Sample path of a particle (the path in green) that moves on the grid according to the Geometric process. From Lemma 5.1, $\delta_{i,B}(B-t)$ can be computed by counting the number of total number of paths from (tn,i) to (Bn,B), and those that lie above the boundary (denoted in red), and taking the appropriate ratio, in the limit $n \to \infty$.

The definition of the vector representation of a path and of path domination is given in Sections 1 and 3, respectively, of Chapter 1 of [12].

The number of lattice paths from $(0, i)$ to (tn, j) staying above the specified boundary is equal to the number of lattice paths from the origin to $(tn, j - i)$ dominated by a path with vector $(a_1 = n(i + 1), a_2 = n(i + 2), \ldots, a_{t-i-1} = n(t - 1), a_{t-i} = nt, a_{t-i+1} = nt, \ldots, a_{j-i} = nt)$. Using this fact, the ratio in (7), $\gamma(i, j, t)(j-i)!/t^{j-i}$, and consequently $\delta_{i,j}(t)$ can be evaluated *exactly*, for any integers i, j, t. A suitably modified definition of $\delta_{i,j}(t)$ for non-integer values of t can be expressed in terms of $\delta_{i,j}(\lfloor t \rfloor)$ and $\delta_{i,j}(\lceil t \rceil)$, and can be computed.

Now, going back to the evaluation of $\mathcal{E}^{c,\text{lower}}_{\text{RR}}$, let $\tilde{r}^L_1$ and $\tilde{r}^{'L}_1$ denote the start and end times of busy periods of a system in which Alice is the only user. Note that providing the attacker with the side information $\mathcal{D}^1_{\text{A}}$ is equivalent to providing him with the start and end times of the busy periods. Now define $F_k = \tilde{r}^{'}_k - \tilde{r}^{'}_{k-1}, k = 2, 3, \ldots$, with $F_1 \doteq \tilde{r}^{'}_1$. The random-variables $\{F_k\}_{k\geq 2}$ are independent and identically distributed random variables (owing to the fact that arrivals from Alice follow a Poisson process which is memoryless), and consequently, the end times of the busy periods form a renewal process. Let P_k denote the set $\{F_l, F_{l+1}, \ldots, F_m\}$ where $l = \arg\max_j\{\tilde{r}^{'}_j < (k-1)c\}$ and $m = \arg\min_j\{\tilde{r}^{'}_j \geq kc\}$. P_k is the set of the busy periods that 'cover' clock period k. It can be shown that the pair (X_k, P_k) form a Markov chain, and also $\mathbf{E}[X_k|\mathcal{D}^1_{\text{A}}] = \mathbf{E}[X_k|P_k]$. Blackwell's celebrated renewal theorem can be used to compute the joint distribution of (X_k, P_k), expressing it in terms of $\delta_{i,j}(t)$. The resulting estimation error can then be computed numerically. The same procedure can be used to compute $\mathcal{E}^{c,\text{upper},\epsilon}_{\text{Priority}}$ as well. Furthermore, it can be shown that in the limit $\epsilon \to 0$, the error incurred by the attacker is same as $\mathcal{E}^{c,\text{lower}}_{\text{RR}}$. This result is true because, as described in Section III-A, through the attack Bob learns the start and end times of the busy periods of the system, for all values of ϵ. When the size of jobs issued by Bob is small, the busy periods of this system are statistically identical

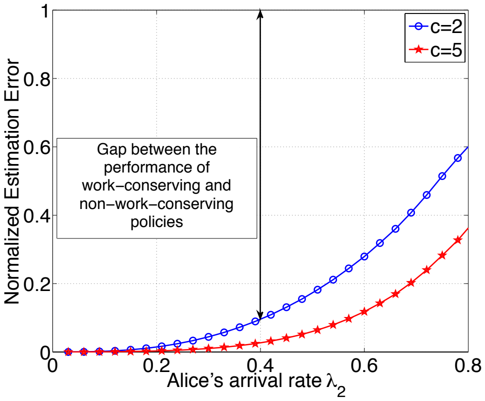


Fig. 5. Plot of $\mathcal{E}_{\mathrm{RR}}^{c,\mathrm{lower}}/\mathcal{E}_{\mathrm{Max}}^{c}$ for the two cases when the clock period is $c = 2$ and $c = 5$. These are also the curves for $\lim_{\epsilon\to 0}\mathcal{E}_{\mathrm{Priority}}^{c,\mathrm{upper},\epsilon}/\mathcal{E}_{\mathrm{Max}}^{c}$. A curve for $\mathcal{E}_{\mathrm{P}}^{c,\lambda}/\mathcal{E}_{\mathrm{Max}}^{c}$ lies below $\mathcal{E}_{\mathrm{RR}}^{c,\mathrm{lower}}/\mathcal{E}_{\mathrm{Max}}^{c}$ for any work-conserving policy P.

to the busy periods of a system where Alice is the only user. The details are skipped for the lack of space. In Figure 5, we plot the computed values of $\mathcal{E}_{\mathrm{RR}}^{c,\mathrm{lower}}$, which is also equal to $\lim_{\epsilon\to 0}\mathcal{E}_{\mathrm{Priority}}^{c,\mathrm{upper},\epsilon}$.

A. Discussion

Recall that $\mathcal{E}_{\mathrm{Priority}}^{c,\lambda}$ is a bound on the performance of all work-conserving policies, and in particular, the privacy offered by round-robin, which is $\mathcal{E}_{\mathrm{RR}}^{c,\lambda}$. $\mathcal{E}_{\mathrm{Priority}}^{c,\mathrm{upper},\epsilon}$ is an upper bound to $\mathcal{E}_{\mathrm{Priority}}^{c,\lambda}$ and $\mathcal{E}_{\mathrm{RR}}^{c,\mathrm{lower}}$ is a lower bound to $\mathcal{E}_{\mathrm{RR}}^{c,\lambda}$. These errors are normalized by $\mathcal{E}_{\mathrm{Max}}^{c}$, the maximum privacy that any policy can offer. A normalized error close to zero means that the policy offers very little privacy, and if it is close to one, the attacker learns no information about Alice's arrival pattern. In Figure 5, we plot $\mathcal{E}_{\mathrm{RR}}^{c,\mathrm{lower}}/\mathcal{E}_{\mathrm{Max}}^{c}$ as a function of λ_2, the arrival rate of jobs from Alice. Note that this is also equal to $\lim_{\epsilon\to 0}\mathcal{E}_{\mathrm{Priority}}^{c,\mathrm{upper},\epsilon}/\mathcal{E}_{\mathrm{Max}}^{c}$, and as a consequence of (4), the curves also represent $\mathcal{E}_{\mathrm{Priority}}^{c,\lambda}/\mathcal{E}_{\mathrm{Max}}^{c}$ and most importantly, $\mathcal{E}_{\mathrm{RR}}^{c,\lambda}/\mathcal{E}_{\mathrm{Max}}^{c}$. In the plot, we consider two scenarios, one where the clock period is set to 2, and the other where it is set to 5. As expected, the attacker incurs a higher normalized error when he wishes to estimate Alice's arrivals with greater precision. The curves represent the maximum estimation error that the best attacker will incur against any work-conserving policy. Note that there is a relatively large gap between the privacy performance of work-conserving and policies that are allowed to idle. For instance, when Alice's arrival rate is less than 0.4, any work-conserving policy can guarantee a privacy no greater than just 10% of the privacy that can be guaranteed by TDMA. In [13], the authors state that in most cloud computing platforms, the load is typically less than 0.2. In such scenarios, the designers of the system need to be aware of the existence and possible exploitation of the timing based side channel discussed in this work. In the 'high-traffic regime', the privacy offered by the round-robin policy is *comparable* to that by TDMA. The reason behind this is the following. As stated in Theorem 4.1, the maximum information that the attacker can learn by performing any attack against the round-robin policy is the start and end times of the busy periods of the scheduling system. When Alice's rate is high, most of the busy periods are of extremely long duration. When busy periods are long, there are several possible arrival patterns of Alice that could lead to the same busy period. Therefore, the attacker learns very little by performing the attack, and therefore incurs a large error. While the curves can be computed for all values of $\lambda_2 < 1$, doing so for higher values of λ_2 requires computation of factorials of large numbers. Owing to the possible numerical errors involved in these computations, we skip plotting the curve in this regime.

VI. Conclusion

In this work, we quantify the privacy offered by work-conserving scheduling policies by showing that the round-robin is a privacy optimal policy in this class, and quantifying its privacy metric. We show that all work-conserving policies fare very poorly on the privacy metric. This is especially true in the low-traffic regime. This is because, when the arrival rate from the user is low, there is typically only one or no jobs from her in the buffer at any given time. Therefore, if the scheduler is forced to serve jobs present in the buffer without idling, the scheduler does not have many options to choose from, and therefore through the process of scheduling, leaks some information about Alice's jobs to the attacker. This observation is consistent with our earlier results in [14] and [15], where we used a correlation based metric to quantify the information leakage. We had observed that although round-robin did leak less information to the attacker compared to FCFS, the two performed similarly in the low traffic regime, and both were equally vulnerable. A surprising result from our analysis is that, even in the medium-traffic regime, there exists no randomized policy which can guarantee a high estimation error for the attacker.

References

[1] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam, "Mitigating timing based information leakage in shared schedulers," in *INFOCOM, 2012 Proceedings IEEE*, pp. 1044 –1052, March 2012.

[2] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP '05, pp. 183–195, IEEE Computer Society, 2005.

[3] N. S. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on tor using long paths," in *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, (Berkeley, CA, USA), pp. 33–50, USENIX Association, 2009.

[4] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis," in *Privacy Enhancing Technologies*, 2012.

[5] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pp. 199–212, ACM, 2009.

[6] K. Zhang and X. Wang, "Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems," in *USENIX Security*, 2009.

[7] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot me if you can: Uncovering spoken phrases in encrypted voip conversations," in *SP '08: Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pp. 35–49, IEEE Computer Society, 2008.

[8] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[9] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam, "Scheduling with privacy constraints," in *2012 IEEE Information Theory Workshop (IEEE ITW 2012), an extended version is available at http://www.ifp.illinois.edu/~kadloor1/kadloor_itw_extended.pdf*, (Lausanne, Switzerland), Sept. 2012.

[10] E. Hahne, "Round-robin scheduling for max-min fairness in data networks," *Selected Areas in Communications, IEEE Journal on*, vol. 9, pp. 1024 –1039, sep 1991.

[11] R. G. Gallager, *Poisson Processes, class notes, available online at http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-262-discrete-stochastic-processes-spring-2011/course-notes/MIT6_262S11_chap02.pdf*.

[12] T. Narayana, *Lattice path combinatorics, with statistical applications*. Mathematical expositions, University of Toronto Press, 1979.

[13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, Apr. 2010.

[14] S. Kadloor, X. Gong, N. Kiyavash, T. Tezcan, and N. Borisov, "A low-cost side channel traffic analysis attack in packet networks," in *IEEE ICC 2010*, 2010.

[15] S. Kadloor, X. Gong, N. Kiyavash, and P. Venkitasubramaniam, "Designing router scheduling policies: A privacy perspective," *Signal Processing, IEEE Transactions on*, vol. 60, pp. 2001 –2012, April 2012.

Appendix A
Proof of Theorem 3.1

The following lemma is used to prove the theorem.

Lemma A.1: Fix an arrival process from Alice. Denote by t_1^n the arrival times of jobs from the attacker and s_1^n be the sizes of these jobs. Let $t_1^{'n}$ be the departure times of these jobs if the scheduler gave priority to jobs of the attacker. For the same set of arrivals from Alice and the attacker, let $\tilde{t}_1^n$ be the departure times of the jobs of the attacker if the scheduler used a policy $\mathrm{P}, \mathrm{P} \in \mathcal{WC}$, where $\mathcal{WC}$ is the class of work-conserving polices. Then, $t_i^{'}$, for each i is a deterministic function of t_1^n, s_1^n and $\tilde{t}_1^n$.

Proof of Lemma A.1: Let $W(u)$ be the total work in the system at time u. Note that $W(u)$ is the same for all non-idling policies [9]. Denote by $\tilde{W}_A(u)$, and $\tilde{W}_B(u)$ respectively, the total work of Alice and Bob in the system at time u when the scheduler policy P. Then $W(u) = \tilde{W}_A(u) + \tilde{W}_B(u)$. Denote by $\langle x \rangle$ the fractional part of a real number x, i.e., $\langle x \rangle \doteq x - \lfloor x \rfloor$.

Claim A.1a: The attacker can compute $\langle W(t_i) \rangle$ for each i.

Proof of Claim A.1a: When the scheduler uses policy P, suppose there are m outstanding jobs from the attacker which have not departed by time t_i. Let $j_1, j_2, \ldots, j_m$ be their indices, i.e., job j_1 is the job from attacker that has arrived by time t_i and departs first after time t_i, j_2 is the second job that departs after time t_i and so on. Suppose $\tilde{t}_{j_1} - t_i \leq s_{j_1}$, then at time t_i, the scheduler should have been busy serving a job from the attacker. In this case, $\tilde{W}_A(u)$ is an integer. Therefore, $\langle W(t_i) \rangle = \langle \tilde{W}_B(t_i) \rangle = \langle \tilde{t}_{j_1} - t_i + \sum_{k=2}^{m} s_{j_k} \rangle$. Now, suppose $\tilde{t}_{j_1} - t_i > s_{j_1}$, then at time t_i, the scheduler should have been busy serving a job from Alice. In this case, the scheduler has to first serve the job from Alice that is in service at time t_i and only then can it move on to serving other jobs. Therefore, job j_1 from the attacker can only depart at time $\langle \tilde{W}_A(t_i) \rangle + q$, for some non-negative integer q. Therefore, $\langle \tilde{W}_A(t_i) \rangle = \langle \tilde{t}_{j_1} - s_{j_1} - t_i \rangle$, and $\tilde{W}_B(t_i) = \sum_{k=1}^{m} s_{j_k}$. Now, $\langle W(t_i) \rangle = \langle \langle \tilde{W}_A(t_i) \rangle + \langle \tilde{W}_B(t_i) \rangle \rangle$, which can clearly be computed by the attacker. ■

To prove the result of the lemma, note that $\forall\ i$,

$$t_{i+1}^{'} = \begin{cases} t_{i+1} + \langle W(t_{i+1}) \rangle + s_{i+1}, & \text{if } t_{i+1} > t_i^{'} \quad (9) \\ t_i^{'} + s_{i+1}, & \text{if } t_{i+1} \leq t_i^{'} , \quad (10) \end{cases}$$

If $t_i^{'} < t_{i+1}$, then the $i + 1^{th}$ job waits only for the service of the job that is already at the server, and then immediately goes into service. Therefore equation (9) follows. If $t_i^{'} > t_{i+1}$, the $i + 1^{th}$ job from the attacker goes into service as soon as the i^{th} job of the attacker gets served. Therefore (10) follows. Also, $t_1^{'} = t_1 + \langle W(t_1) \rangle + s_1$. From Claim A.1a, $\langle W(t_i) \rangle$ can be computed by the attacker for each i, and consequently $t_1^{'n}$. ■

Proof of Theorem 3.1: From Lemma A.1, for any work-conserving policy P used by the scheduler, the attacker can always simulate the observations which he would make if the scheduling policy were a priority policy. Denote by $\mathcal{E}_{\text{Priority}}^{c,\lambda}$ the estimation error incurred by the strongest attacker against the priority policy. Using the same notation from Lemma A.1, for every $\mathrm{P} \in \mathcal{WC}$, we then have the following:

$$\begin{aligned} & \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k | t_1^n, s_1^n, \tilde{t}_1^n]\right)^2\right] \\ & \quad = \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k | t_1^n, s_1^n, \tilde{t}_1^n, t_1^{'n}]\right)^2\right] \qquad (11) \\ & \quad \leq \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k | t_1^n, s_1^n, t_1^{'n}]\right)^2\right], \qquad (12) \end{aligned}$$

where, (11) follows from Lemma A.1, and (12) follows from an elementary result from estimation theory which states that discarding information leads to an inferior estimate. Therefore,

$$\begin{aligned} & \min_{t_1^n, s_1^n : \frac{\sum_{j=1}^{n} s_j}{Nc} < \lambda} \frac{1}{N} \sum_{k=1}^{N} \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k | t_1^n, s_1^n, \tilde{t}_1^n]\right)^2\right] \\ & \leq \min_{t_1^n, s_1^n : \frac{\sum_{j=1}^{n} s_j}{Nc} < \lambda} \frac{1}{N} \sum_{k=1}^{N} \mathbf{E}\left[\left(X_k - \mathbf{E}[X_k | t_1^n, s_1^n, t_1^{'n}]\right)^2\right], \end{aligned}$$

and consequently, $\mathcal{E}_{\mathrm{P}}^{c,\lambda} \leq \mathcal{E}_{\text{Priority}}^{c,\lambda}$, $\forall \mathrm{P} \in \mathcal{WC}$. ■

Appendix B
Proof of Lemma 3.2

For some j, if $t_j^{'} = t_j + \epsilon$, i.e., if the j^{th} job from the attacker goes into service immediately upon its arrival, then the system must be empty at time t_j. Hence, t_j marks the start of a busy period that is initiated by an attacker's job. On the other hand, if $t_j^{'} > t_j + \epsilon$, and $t_{j-1}^{'} < t_j^{'} - 1 - \epsilon$, then, $t_j^{'} - 1 - \epsilon$ marks the start of a busy period that is initiated by a job from Alice. Because every busy period is initiated by either a job from Alice or the attacker, and because the events that are described above occur only at the start of busy periods, the start times of all the busy periods can be computed by the attacker, and he can furthermore figure out if these busy periods are initiated by an attacker's job or

Alice's job. To compute the end times of the busy periods, note that the maximum time the scheduler stays idle between two consecutive busy periods is less than 1. This is a consequence of the arrival process from the attacker. Furthermore, note that all the busy periods end with the departure of a job from the attacker irrespective of whether it was initiated by a job from Alice or the attacker. Therefore, the last departure before the start of the next busy period marks the end of the current busy period.

APPENDIX C
PROOF OF LEMMA 3.3

Recall that all of the busy periods end with the service of a job from the attacker, and in each busy period, jobs from the attacker and Alice are served alternately. Hence, all the busy periods that are initiated by a job from the attacker are of duration $(k+1)\epsilon + k$, for some non-negative integer k. Furthermore, preceding the busy period initiated by the attacker's job, there is no arrival from either of the users for a duration of 1 time unit. On the other hand, all busy periods initiated by a job from Alice are of duration $(k+1)\epsilon + k + 1$, for some non-negative integer k. Preceding such a busy period is a period of duration less than 1 where there are no arrivals from either of the users. Therefore, given the end times of the busy period, the attacker can infer the duration of the busy period and also if the busy period was initiated by an attacker's job or Alice's job. Therefore, if the l^{th} busy period was initiated by an attacker's job, departures in that busy period occur at times $r_l + \epsilon, r_l + 1 + 2\epsilon, \ldots, r^{'}_l$. If the l^{th} busy period was instead initiated by a job from Alice, then the departures in that busy period occur at times $r_l + 1 + \epsilon, r_l + 2 + 2\epsilon, \ldots, r^{'}_l$. Therefore, the departure times of all of attacker's jobs can be computed knowing the start and end times of the busy periods. Also, because the arrival and departure times of the jobs issued by the attacker are related by (3), the arrival times of the jobs can be computed as well.

APPENDIX D
PROOFS OF LEMMAS 4.2 AND 4.3

A. *Proof of Lemma 4.2*

We will use induction to prove this lemma. First, note that $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^1_{\mathrm{A}}(u)\ \forall u \in (0, t_1)$. This is because the two systems have the same arrivals till then. At time t_1, the scheduler is either busy serving a job from Alice, or is idle. If it is busy, then the scheduler waits till it completes the service of this job, and then switches over to serve Bob. In either case, Bob's incoming job goes into service at time $\tilde{t}_1 = \inf\{u > t_1 : \mathcal{D}^1_{\mathrm{A}}(u) = \lceil \mathcal{D}^1_{\mathrm{A}}(t_1) \rceil\}$, and the job departs the system at time $t^{'}_1 = \tilde{t}_1 + s_1$. Therefore, $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^1_{\mathrm{A}}(u), \forall u \in (0, \tilde{t}_1)$ and because Alice does not get any service when the scheduler serves Bob, $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^1_{\mathrm{A}}(\tilde{t}_1), \forall u \in (\tilde{t}_1, t^{'}_1)$.

Statement of the induction: Given arrival times of the first k jobs from Bob, t_1^k, their sizes s_1^k, departure times of the first $k-1$ of his jobs, $t_1^{'(k-1)}$, $\mathcal{D}^1_{\mathrm{A}}$, and $\mathcal{D}^2_{\mathrm{A}}(u), \forall u \in (0, t^{'}_{k-1})$, the departure time of the k^{th} job, $t^{'}_k$, and $\mathcal{D}^2_{\mathrm{A}}(u), \forall u \in (0, t^{'}_k)$ can be computed.

Proof of induction: The base case for induction is already proved. We need to prove it for some $k > 1$ assuming it is true for all times before that. The arrival time of the k^{th} job from Bob falls into one of the following cases:

Case 1: $t_k < t^{'}_{k-1}$. Note that $\mathcal{D}^2_{\mathrm{A}}(u) \leq \mathcal{D}^1_{\mathrm{A}}(u), \forall u$. This is because in the second system, there are jobs from the attacker along with the jobs from Alice. Also, $\dot{\mathcal{D}}^2_{\mathrm{A}}(t^{'-}_{k-1}) = 0$, where $t^{'-}_{k-1}$ is an infinitesimal small time before $t^{'}_{k-1}$. Therefore, if $\mathcal{D}^2_{\mathrm{A}}(t^{'}_{k-1}) = \mathcal{D}^1_{\mathrm{A}}(t^{'}_{k-1})$, then it must be the case that $\dot{\mathcal{D}}^1_{\mathrm{A}}(t^{'}_{k-1}) = 0$, implying that all of Alice's jobs that arrived before $t^{'}_{k-1}$ have been served by then. Therefore, the k^{th} job from Bob goes into service immediately and departs the system at time $t^{'}_k = t^{'}_{k-1} + s_k$. In this case, $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^2_{\mathrm{A}}(t^{'}_{k-1}) \forall u \in (t^{'}_{k-1}, t^{'}_k)$.

If $\mathcal{D}^2_{\mathrm{A}}(t^{'}_{k-1}) < \mathcal{D}^1_{\mathrm{A}}(t^{'}_{k-1})$, at time $t^{'}_{k-1}$, there is at least one unserved job from Alice in the system, which goes into service at time $t^{'}_{k-1}$. Therefore, $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^2_{\mathrm{A}}(t^{'}_{k-1}) + u - t^{'}_{k-1}, \forall u \in (t^{'}_{k-1}, t^{'}_{k-1} + 1)$, and $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^2_{\mathrm{A}}(t^{'}_{k-1}), \forall u \in (t^{'}_{k-1} + 1, t^{'}_{k-1} + 1 + s_k)$. $t^{'}_k = t^{'}_{k-1} + 1 + s_k$.

Case 2: $t_k > t^{'}_{k-1}$. In this case, after serving the $k - 1^{th}$ job from Bob, the scheduler switches over to Alice and serves her jobs back to back (if there are jobs to be served) till the k^{th} job from Bob arrives. Therefore $\forall u \in (t^{'}_{k-1}, t_k)$,

$$\mathcal{D}^2_{\mathrm{A}}(u) = \min\{\mathcal{D}^2_{\mathrm{A}}(t^{'}_{k-1}) + u - t^{'}_{k-1}, \mathcal{D}^1_{\mathrm{A}}(u)\}.$$

The time when the k^{th} job from Bob goes into service is given by $\tilde{t}_k = \inf\{u > t_k : \mathcal{D}^2_{\mathrm{A}}(t_k) + u - t_k = \lceil \mathcal{D}^2_{\mathrm{A}}(t_k) \rceil\}$. Then, $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^2_{\mathrm{A}}(t_k) + u - t_k, \forall u \in (t_k, \tilde{t}_k)$, $t^{'}_k = \tilde{t}_k + s_k$, and $\mathcal{D}^2_{\mathrm{A}}(u) = \mathcal{D}^2_{\mathrm{A}}(\tilde{t}_k), \forall u \in (\tilde{t}_k, t^{'}_k)$.

B. *Proof of Lemma 4.3*

The available information to the attacker when he issues his second job is no more than the time when he issued his first job, its size, and its departure time. Therefore, by the result of the Lemma 4.2, t_2 is a function of t_1, s_1 and $\mathcal{D}^1_{\mathrm{A}}$. By a similar argument, t_3 is dependent at most on $t_1, t_2, s_1, s_2, t^{'}_1$ and $t^{'}_2$, all of which are just a function of t_1, s_1 and $\mathcal{D}^1_{\mathrm{A}}$, and so on. Before issuing his first job, the attacker has no information about Alice's arrivals. Hence t_1 and s_1 are independent of any function of the arrival times of Alice's jobs, in particular, X_k.

A Timing Channel Spyware for the CSMA/CA Protocol

Negar Kiyavash, *Member, IEEE,*[*], Farinaz Koushanfar, *Member, IEEE*[†], Todd Coleman, *Member, IEEE*[‡], and Mavis Rodrigues, *Student Member, IEEE,*[*]
[*] Coordinated Science Laboratory, University of Illinois Urbana-Champaign, Urbana, IL
{kiyavash, mrodrig5}@illinois.edu
[†]Electrical and Computer Engineering, Rice University, Houston, TX
{farinaz}@rice.edu
[‡] Department of Bioengineering, University of California, San Diego, CA
{tpcoleman}@ucsd.edu

***Abstract*—This paper presents the design and implementation of spyware communication circuits built into the widely used Carrier Sense Multiple Access with collision avoidance (CSMA/CA) protocol. The spyware components are embedded within the sequential and combinational communication circuit structure during synthesis, rendering the distinction or dissociation of the spyware from the original circuit impossible. We take advantage of the timing channel resulting from transmission of packets to implement a new practical coding scheme that covertly transfers the spied data. Our codes are robust against the CSMA/CA's random retransmission time for collision avoidance and in fact take advantage of it to disguise the covert communication. The data snooping may be sporadically triggered, either externally or internally. The occasional trigger and the real-time traffic's variability make the spyware timing covert channel detection a challenge. The spyware is implemented and tested on a widely used open-source wireless CSMA/CA radio platform. We identify the following performance metrics and evaluate them on our architecture: 1) efficiency of implementation of the encoder; 2) robustness of the communication scheme to heterogeneous CSMA/CA effects; and 3) difficulty of covert channel detection. We evaluate criterion 1) completely theoretically. Criterion 2) is evaluated by simulating a wireless CSMA/CA architecture and testing the robustness of the decoder in different heterogeneous wireless conditions. Criterion 3) is confirmed experimentally using the state-of-the-art covert timing channel detection methods.**

I. Introduction

Rapid technological advances in wireless communication and the growth in the number and diversity of applications and services raise the demand for data-integrity and communication security. The complexity of the emerging applications and platforms introduce newer vulnerabilities that need to be carefully identified, analyzed, and addressed.

The emerging 802.xx standard protocols must be continually enriched with security features to meet the new and more critical application demands. For instance, the IEEE 802.22 is a cognitive radio standard being developed to bring broadband access to less populated rural areas by using vacant TV channels. To address the complex adaptive nature required by the technology, a cognitive radio device is typically implemented by a general purpose computer processor that also runs radio application software. As a result, it is susceptible to spyware and malicious software or hardware.

In this paper, we present a new transparent and robust covert communication scheme that can reliably spy the chip data to outside entities by exploiting the timing channel resulting from inter-arrival times of the legitimate transmitted packets. Our scheme can be embedded within any medium access control (MAC) protocol that avoids collision in packet retransmissions by using an exponential back-off rule. For instance, the IEEE 802.11 Wireless Local Area Network (WLAN) standard's main MAC layer protocol, Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), is an example of a widely used protocol that employs such a rule.

The effects of the CSMA/CA's back-off rule perturbs packet timings by virtue of the queuing and hence introduces a non-standard noisy channel that interferes with timing-based communication purposes. Although the maximum rate of information exchange of such channels (i.e., capacity) was characterized in [1], only recently have practical encoding/decoding methods to instantiate the theoretical limit with low (i.e., linear) complexity been introduced [2]. However, to date, there have been no practical *covert* information encoding methods with low encoder/decoder complexity to instantiate the findings of [1]. Here, we establish the first reliable covert timing channel that is also robust to the effects of CSMA/CA's back-off rule.

We identify the following performance metrics and evaluate them on our architecture:

1) Transparency of the spyware circuitry: This criterion pertains to the efficiency of implementation at the encoder to prevent the detection of the spyware circuitry.
2) Robustness: This criterion tests the robustness of the communication scheme to heterogeneous CSMA/CA effects which act as noise.
3) Transparency of the covert channel: This criterion measures the difficulty of the covert channel detection.

We evaluate criterion 1) completely theoretically. Criterion 2) is evaluated by simulating a wireless CSMA/CA architecture and testing the robustness of the decoder in different heterogeneous wireless conditions. Criterion 3) is confirmed

This work was supported in part by AFOSR under Grants FA9550-11-1-0016, 727 AF Sub TX 0200-07UI, and FA9550-10-1-0573, and NSF CCF 10-54937 CAR and CCF 10-65022.

experimentally using state-of-the-art covert timing channel detection methods.

The main contributions of this paper are as follows:

- We present the first design of spyware integrated within the communication circuitry of wireless CSMA/CA that exploits the timing channel resulting from inter-arrival times of packets to leak data from the chip.
- We introduce a *covert* channel encoding that that utilizes the coding methodology developed in [2]. This methodology allows us to encode timing information in presence of the worst non-negative additive noise, the exponential noise. [3].
- Our spyware employs a low-complexity error-correcting code framework that is robust to timing perturbations resulting from the CSMA/CA's collision avoidance back-off strategy. This back-off strategy introduces a non-standard queuing noisy channel that interferes with timing-based communication purposes.
- We show the difficulty of the spyware detection, both at the hardware level and at the packet timing level, from both theoretical and practical perspectives.
- To substantiate our theoretical findings and to evaluate the overhead, the spyware is implemented and tested on the widely used WARP wireless radio platform [4].

The remainder of the paper is organized as follows. The related literature is surveyed in Section II. The preliminaries are described in Section III. Section IV introduces our theoretical results for covert channel embedding. In Section V, we devise the new spyware embedding algorithm. Detection of the spyware in circuitry and at the communication channel is presented in Section VI. Comprehensive experimental results for implementation, simulations, and detection are demonstrated in Section VII. We conclude in Section VIII.

II. Related Work

Research in hardware malware has recently emerged, especially after the DoD's Defense Science Board comprehensive report on the subject in 2005 [5]. Much of this work has concentrated on detection of foundry-inserted Trojans (malware), which is typically done by invasive or noninvasive study of the post-silicon ICs and comparison with the original design files to detect the modifications to the manufactured chips [6]. Note that the existing hardware Trojan detection mechanisms are not able to detect our spyware because our assumptions are radically different (we will elaborate on this issue in Section VI). A number of methods for protection of ICs and third-party IP cores against piracy have been invented [7]. The threat of insertion of spyware in communication and security hardware modules has been identified [8], [9]. King et al. [10] have designed and implemented malicious hardware that can get hidden login access or can steal the security keys on a microprocessor. The malicious hardware is embedded at the micro-architecture level by addition of a block of gates and does not modify the internals of the architectural blocks. The key advantage of our pre-synthesis spyware integration is that it is not distinguishable/removable from the radio's internal structure implementing the protocol's state-transitions.

Historically, timing channels are synonymous with covert channels [11]–[15]. Covert channels are mechanisms for communicating information in ways that are difficult to detect. While the focus of earlier work has been mainly on disrupting or completely eliminating covert timing channels [13], [16], the recent work has focused more on detection of covert timing channels [17]–[19]. Moreover, timing side channels pose formidable security and privacy threats [20]–[22].

III. Background

In this section, we first provide a brief background on the employment of the CSMA/CA protocol in the widely used IEEE Wireless Standard 802.11. Afterwards, we summarize the methods for triggering the spyware. Lastly, the incentives for designer spyware insertion are discussed.

A. CSMA/CA in IEEE 802.11

The MAC layer of the IEEE Standard 802.11 for Wireless Local Access Networks (WLANs) uses CSMA/CA for collision avoidance [23].

The main idea behind the CSMA/CA collision avoidance strategy is that the channel is always sensed before any transmission. If the channel is sensed busy, the sender waits until the channel is idle and then goes through a random back-off period before retrying. The random back-off period ensures fairness among contending transmissions. More precisely, the transmission occurs only if, during a fixed period of time, equal to a distributed interface state (DIFS), the channel is sensed idle. If the sensed channel is busy during or immediately after the DIFS, the station continues monitoring the channel until the channel is sensed idle for a DIFS period. At this time, the CSMA/CA generates a random back-off interval before transmitting to avoid the possible collisions by minimizing the probability of collision with packets transmitted by other radios. Even if the medium is sensed idle during the DIFS period, a radio waits a random back-off time between consecutive transmission of two packets to avoid channel capture. The flowchart in Figure 1 shows the basic steps of the CSMA/CA at the transmitting radio.

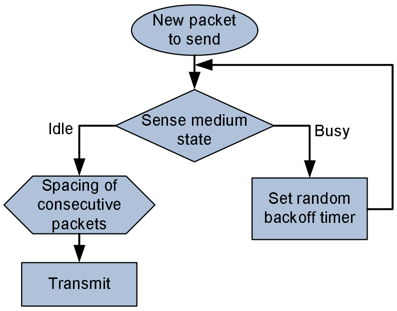


Fig. 1. Flow of the CSMA/CA packet transmission with exponential back-off time.

Moreover for each packet transmission, an exponential back-off method is used and to signal a successful packet transmission, an ACK is transmitted by the destination radio. This ACK is automatically transmitted at the end of the

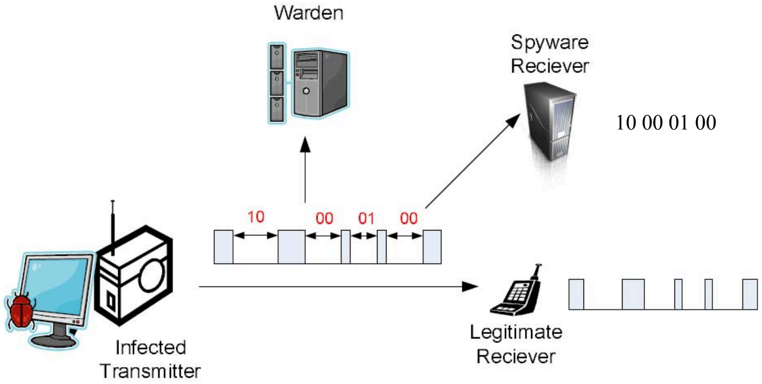


Fig. 2. Covert communication: A warden inspects the communication, but is unable to decode messages modulated by the packet timings.

packet, after a short-time inter-frame space (SIFS). Since the combination of the SIFS and channel delay is shorter than DIFS, the channel cannot be detected idle for a DIFS until the end of an ACK.

B. Triggering the Spyware

The spyware is activated upon the arrival of a trigger, which may come from either internal or external sources. The spyware does not need to be active all the time. We call the active duration of a spyware the *spying interval*. An internal spyware trigger comes from within the hardware. The two most obvious choices for an internal incitement are the states of the registers in the design, and the clock. The states of the register can be utilized in a number of ways - for example, by arriving at a certain internal state of the communication controller, or upon reaching a certain counter state. Recent research has suggested the possibility of designing challenging (hard-to-detect) Trojans and triggers [24]. Since we implement the trigger at the hardware level and integrate it within the design, it is impossible to detect these types of signals by studying the radio output.

The external triggers have to find a way for the outside source to reach the spyware. The most common method is via the communication channel. Note that other hardware interfaces to the outer world, such as sensors or power lines, can be used as extrinsic stimuli. The covert communication channel can also be used for signaling the trigger. For example, the radio can be intentionally kept busy for certain time intervals. An important observation is that the external trigger does not have to be sent via the covert channel. For example, the receiver can be manipulated such that a certain combination of the input signals would not create an interrupt to the layers above. At the same time, the carrier sense signal can be raised high by the spyware, disallowing the legitimate messages to route via the communication channel. Unless the secret trigger combination is known and the radio is under physical tests observing its activity, detecting this type of external trigger is extremely difficult, if not impossible. Note that the cost of triggering can be negligible. For example, very often a wire or a single gate is used for triggering [25].

C. Incentives for Designer Spyware

Application Specific Integrated Circuit (ASIC) is the predominant technology used for designing power-efficient and low-cost communication circuitry. At least three parties are involved in a typical ASIC supply chain scenario: a design company, a third party fabrication house, and the end user. The spyware (Trojan) discussed in this paper is inserted by the design company during the circuit synthesis which could be applied to both ASIC and Field-Programmable Logic Array (FPGA) technologies. Note that the inclusion of designer spyware is not limited to the untrusted companies for malicious purposes. Contemporary designers often use the backdoors implemented by the spyware for digital rights management purposes.

Several research and development efforts in the area of hardware security have focused on the problem of hardware Trojan where the treat model has been post-synthesis Trojan insertion by the ASIC fabrication house [25], [26]. Since post-synthesis modification of the design blueprint is prohibitively expensive, the fabrication house could only insert the added Trojan components within the white spaces on the blueprint; the white spaces are unavoidable since they are a side-product of automatic synthesis and layout processes. This attack could have the same impact on the communication packets as the designer's spyware. However, this attack may be detectable by the design company who has access to the original design specifications post-synthesis. The designer could use the side-channel tests to compare the measurements from the fabricated chip with the original design simulations [27].

Side-channel tests are inapplicable for the designer spyware detection discussed in this paper. Our spyware is inserted during the synthesis, while the Trojan-free pre-synthesis specifications are never shared with other parties in the IC supply chain. The spyware and its trigger can be designed to be point functions, and thus they can be efficiently obfuscated (hidden) within the exponential state-space of the design [7], [24], [28]. As long as the circuit performs the intended input/output tasks, the hidden functionality of the spyware cannot be distinguished post-synthesis.

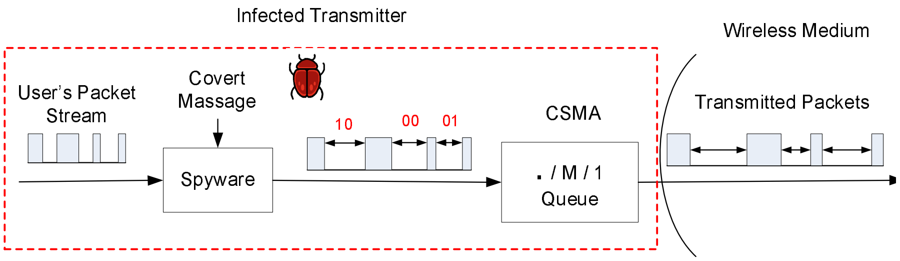


Fig. 3. Bits are encoded into the arrival times of the packet sequence originating from the user's packet stream. The packets are sent over the wireless medium with the augmented inter-arrival times.

IV. Covert Communication with Timings

Historically, timing channels are synonymous with covert channels [11]–[15], [17], [18], [29]. Covert channels are mechanisms for communicating information in ways that are difficult to detect. Packet networks are designed with the goal of communicating through packet contents and their headers and not the timings. Hence, the timing channel induced by the inter packet timings provides a side channel that can be utilized for covert communications. Figure 2 illustrates a covert timing channel between an infected transmitter and a legitimate receiver. The *warden* (a.k.a eavesdropper) sees the exchange of packets but fails to realize the covert communication in packet timings. On the other hand, the spyware receiver, which has side information pertaining to the parameters of the encoding scheme, (see Section IV-B) can decode the message conveyed through timings.

As mentioned in Section III, the CSMA/CA protocol results in random delays in the packet inter-arrival times. We will model the impact of the CSMA/CA as a first-come first-served (FCFS) queuing system. In particular, a special case of a FCFS queuing timing channel is the "exponential server timing channel (ESTC)", where service times are independent and identically distributed (i.i.d.), and memoryless: in continuous-time, they are exponentially distributed of rate μ - with probability density function (PDF) $P_{S_i}(s) = \mu e^{-\mu s}$, and in discrete-time they, are geometrically distributed of rate μ - with probability mass function (PMF) $P_{S_i}(s) = \mu(1-\mu)^{s-1}$. In either case, by defining a_i (d_i) to be the time of the ith packet arrival (departure) of the queue, we have that:

$$P\left(\underline{d}^n|\underline{a}^n\right) = \begin{cases} \prod_{i=1}^n P_{S_i}(s) & \text{if } d_i \geq a_i, \ \forall\, i \in \{1,\ldots,n\} \\ 0 & \text{otherwise} \end{cases} \tag{1a}$$

$$s_i = d_i - \max(a_i, d_{i-1}) \tag{1b}$$

Ideally, in the absence of any queuing, the information in the timings between successive packets could be used for unlimited rates of communication. However, by adding random delays to the inter-arrival times of the packets, the CSMA/CA protocol acts as "noise" to the packet timings, as illustrated in Figure 3. It is the job of the spyware receiver to decode the transmitted message by virtue of the departure process of the queue. The ESTC has the smallest capacity among all FCFS queuing channels with the same average service rate [30], hence a covert communication scheme that can survive the ESTC is most likely robust to other types of queuing noise.

The maximum rate of communication for an ESTC with an arrival process of rate λ satisfies [1]

$$C(\lambda) = \lambda \log_2 \frac{\mu}{\lambda}, \ \ \lambda < \mu, \ \ \text{bits/s} \tag{2}$$

where the maximum of (2) is achieved with an input Poisson process at a rate $\lambda^* = e^{-1}\mu$. However it was not clear until recently [2] how to develop practical codes that can withstand the noise from the queueing channels and approach the fundamental limits given by (2). In this work, we will for the first time use similar code constructions for the purpose of *covert* communication over a timing channel introduced due to queuing effects of CSMA/CA.

A. Encoder Structure

The coding schemes of [2] involves an algebraic code construction that allows for a simple encoder, thus lending itself nicely to the design of an in-circuit spyware. In short, a coset code construction [31] followed by shaping is used [31]–[34]. Specifically, we first use a sparse graph coset code (H, s):

$$\mathcal{C} = \{x : Hx = s\}$$

where H is an m by n matrix, s is a length-m vector, and all algebraic operations are defined over the finite field $\mathbb{F}_Q$. The encoder module uses the generator-representation of the code, where a length-k ($k = n - m$) information vector is mapped to a length-n codeword vector x by use of an n by k generator matrix G and a length-n vector p:

$$x = Gu + p.$$

Thus, in order to perform these operations, only standard finite field arithmetic is required. Lastly, a "shaping" operation is performed to map each $x_i \in \mathbb{F}_Q$ to an inter-arrival time $z_i \in \mathbb{R}_+$ using a table-lookup:

$$z_i = B_i(x_i).$$

This idea exploits the well-known result [31] that the ensemble of random linear coset codes produces i.i.d. code symbols x_i uniformly distributed over $\mathbb{F}_Q$. So B_i is chosen to map a uniformly distributed random variable, $x_i \in \mathbb{F}_Q$, to a non-uniform, exponentially distributed, inter-arrival time $z_i \in \mathbb{R}_+$. This can be done via the use of a "dither" D_i, used widely in quantization for provably good performance [35], and the inverse CDF $F_Z^{-1}(\cdot)$ of an exponentially distributed random variable:

$$U_i(x) = \left[\frac{\mathcal{R}(x)}{Q} + D_i\right]_{\text{mod } 1}, \tag{3a}$$

$$B_i(x) = F_Z^{-1}(U_i(x)) \tag{3b}$$

where $\mathcal{R}(x)$ denotes interpreting $x \in \mathbb{F}_Q$ as a member of $\mathbb{R}_+$. Note that the details of the dither are immaterial to the encoder/decoder pair: B_i is simply a length-Q vector.

B. Decoder Structure

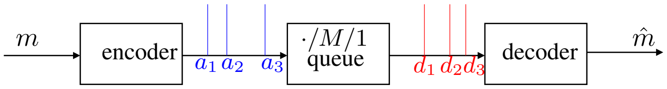


Fig. 4. A block diagram of the spyware encoder and decoder. The spyware receiver decodes the message using the packet's timing sequence (red).

The coding scheme of [2] exploits the graphical structure of the probabilistic dynamics of a queuing system (1a) that enables a low-complexity message-passing decoder. The decoder module resides in software at the spyware receiver, and upon receiving the packets, it decodes the transmitted message using the inter-packet times of the departure process (as depicted in Figure 4).

In this scheme, the concatenation of the graphical model of the error-correcting code (specified by the sparse matrix H, the syndrome s, the shaping operator B_i), and the probabilistic dynamics of a queuing system lead to an aggregate graphical model that is highly structured. This enables the use of standard low-complexity message-passing algorithms (belief propagation, also termed the sum-product algorithm) as a viable decoding solution that produces low bit-error rates even close to the capacity.

V. Implementation

The CSMA/CA which constitutes the communications MAC layer may be implemented in software or hardware. Either way, The MAC needs to interact with the physical layer (PHY) and the hardware components that provide the necessary information for the correct operation of the protocol. The PHY layer can be modified without affecting the upper layer MAC or notifying the changes to any of the higher layers, or even providing false information to them. Figure 5 shows different ways of interaction between the MAC and the PHY layer. The MAC queries the PHY for the status of the medium and timing information. As can be seen on the figure, the PHY can be modified to mislead the MAC by sending false timing information to it, or even notify the MAC that the medium is busy when it is idle. Perhaps the most important mean for spying the information at this layer is that the front transmission and reception of the packets must be done through the PHY.

In our implementation of the spyware, we modify the PHY so that the timing of the sent packets are altered in order to covertly send the stolen data to the outside world. To achieve this goal, we implement the encoder explained in Section IV-A. The block diagram of the encoder circuit is shown in Figure 6. The encoder circuit was implemented on a reconfigurable radio platform that implements a programmable baseband architecture. We used the WARP platform developed at Rice University for our implementation [4]. This programmable baseband radio contains an FPGA board that can be reconfigured. An advantage of the FPGA implemented in WARP (compared to the simple FPGAs with only lookup tables) is that there it also contains dedicated adders and multipliers that facilitate implementation of various DSP and coding algorithms.

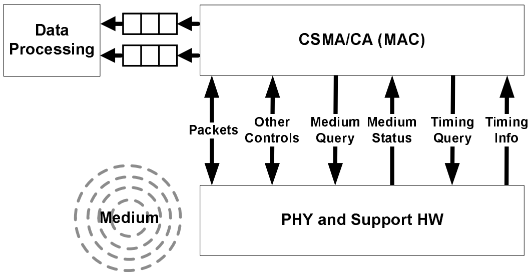


Fig. 5. Interaction between CSMA/CA and the PHY.

By programming the FPGA, both the generator matrix G and the length-n vector p are hard coded in the PHY layer. G is generated for a specific finite field $\mathbb{F}_Q$. Algorithm 1 shows the steps for constructing the circuit representing multiplications by G. First, a template table T is constructed for $\mathbb{F}_Q$. T has Q entries representing small circuits that perform addition and multiplication of constants from 0 to $Q-1$ with a variable in $\mathbb{F}_Q$. The circuits in the template table are simple XOR gates, constants, or wires. The circuit is generated by the loop in Lines 5 to 7 in the algorithm.

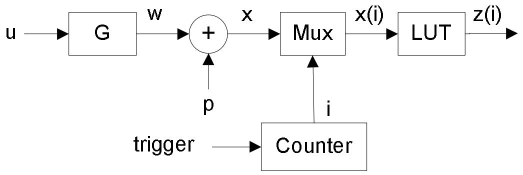


Fig. 6. Block diagram of the encoder circuit.

The final circuit is a network of XOR gates. The output of this circuit is then added to the constant vector p. This step is also hard coded in the circuit by inverting some of the bits of w to generate x. Each element in x represents an index for a specific inter-arrival time value stored in the LUT. The elements of x are traversed using a counter. The counter is reset when the trigger is raised. Every time a packet is successfully transmitted, the counter is incremented for generating the next inter-arrival time.

VI. Spyware Detection

The spyware can be distinguished either by realizing the presence of the spyware circuitry on the chip or by detecting the covert timing channel. We next show that the neither of these two measures are feasible.

A. Detection of Spyware Circuitry

It should be noted that the circuitry of Section V is interwoven and integrated within the physical layer functions. A High Level Description (HDL) Hardware language is used for programming the boards. The HDL code is then compiled and synthesized to hardware components and automatically

Algorithm 1 Algorithm for Encoding with the Generator matrix G

```
Require: $Q$: the field size to be used, $G$: $n \times k$ matrix over $\mathbb{F}_Q$, T: $Q \times Q$ matrix over $\mathbb{F}_Q$.
Input: $u$, the input vector to the circuit of size $k$
Output: $w$, the output vector of the circuit of size $n$
for $i = 1$ to $n$ do
    $W(i)$=0
    for $j = 1$ to $k$ do
        $W(i)$=$W(i) \oplus T(G(i,j), u(j))$
    end for
end for
```

placed on the circuit. We note that the circuit elements used for realizing the algorithm and blocks shown in the previous section are mostly combinational. There are very few sequential elements that use the LUTs, for example the counters. The overhead of the encoder is a function of the size of G. While a smaller G provides a smaller overhead, there is a tradeoff between the size of G and the accuracy of decoding of the encoded message. An example implementation of the encoder and the tradeoff between the overhead and accuracy of the decoder is shown in Table I of Section VII-A. Note that detection of spyware by the power consumption is not feasible since its impact can be easily obfuscated, since the spyware circuit can be actively consuming power without sending any information out. Therefore, it cannot be used for spyware detection purposes. The specific power consumption number is very dependent on the implementation. In our simulated implementation, it was less than 0.5% when the trigger was constantly on. This number is below the noise floor for the measurable power from the supply pin of WARP boards and is not detectable.

The class of attacks that are addressed in this paper are channel-based, where the communication packet timings can be probed. This is because of our spyware implementation scenario. The designer is the implementer of the covert channel and therefore, others who are users would only utilize the implementation. We believe that this is a reasonable assumption for at least three reasons. First, for many products, the manufacturers may provide a backdoor to the design that would often remain unnoticed. The method provided in this paper can be used as a possible backdoor for the implementer. The reason is that the typical users are not provided with the blueprint of the circuitry to be able to investigate the designs as those are often proprietary information and trade secrets. There have been notable cases where it was revealed that the manufacturer has intentionally implemented a backdoor to the design, so this attack is realistic and has been reported. Second, after packaging, the internals of many electronic products are clandestine to the users and cannot be easily debugged and tested. It is true that in our implementation, we use a programmable platform to evaluate our method. One disadvantage or reconfigurability might be that it can be easily reprogrammed and maybe even probed to a certain degree. However, we emphasize that in real products there is a possibility of implementing such channels in ASIC which cannot be reprogrammed or removed. Third, the fabrication house attack scenarios are drastically different than those in our paper as discussed in Section III-C.

Let us also discuss a scenario in which another entity designs a circuit with exactly the same functionality. Even this circuit would not provide a good comparison point, because of the degree of freedom in selecting the design components. In other words, there is no certain measure for the expected power or timing that can be computed for a given high level specification, so there would be little relevance between the new entity's design and the original circuitry without the spyware. Thus, the circuit-level attack is not applicable to our spyware scenario.

B. Detection of Covert Communication

The countermeasures to covert timing channels come mainly in two flavors: detection and disruption. The detection of covert timing channels is based on statistical tests that can distinguish between legitimate and covert traffic. The disruption (e.g., jamming) of covert timing channels is done by an active warden that aims at preventing the covert communication usually with the undesired side effect of degraded system performance. While the focus of earlier work has been mainly on disrupting or completely eliminating covert timing channels [13], [16], the recent work has focused more on detection of covert timing channels [17], [18]. This is mainly because the disruption of timing channels is usually costly (either in terms of resources used to active warden or the degradation of the system performance). One possible disruption is random delaying of the traffic, which reduces the performance (e.g. by reducing the capacity or increasing the error rates) of covert timing channels. Notably, our wireless timing channel is susceptible to injection of false packet events by an external entity. For instance, a second node that transmits cover traffic claiming the identity of the node containing the spyware. However, such a countermeasure is both costly in terms of the attacker resources and the lost capacity of the legitimate users of the channel. Therefore, it is highly desirable to employ disruption mechanisms only after one has verified the presence of a covert channel reliably. Moreover, often cryptographic measures can deal with such scenarios. Specifically in our case once the presence of spyware deduced covert channels is detected, the warden will eliminate the covert channel by eradicating the spyware.

We check the detection resistance of our scheme against the current state-of-the-art covert timing channel detection tests.

- **Kolmogorov-Smirnov (K-S) Test.** The KS-test is a non-parametric and distribution free test that evaluates the equality of one-dimensional distributions of probability. The objective is to determine if two datasets (or one dataset and a reference distribution) differ significantly. For the purpose of detection of covert channels, *legitimate traffic* (in the absence of covert communication) is compared to the data collected in a situation where it is suspected that a covert communication is happening. We shall call the samples from the later situation as *test traffic*. If the test traffic is sufficiently different from the legitimate traffic, it is declared that a covert channel has been detected. The KS-test measures the maximum distance between the empirical cumulative distribution functions (CDF) of the legitimate and the test traffic,

$$\max_x |F_T(x) - F_L(x)|, \tag{4}$$

where $F_T(x)$ and $F_L(x)$ denote the empirical CDFs of the test and legitimate traffic.

- **Regularity Tests.** Regularity tests are based on the assumption that the covert traffic is more regular than the legitimate traffic. One example of such a test is Cabuk et al.'s heuristic regularity test [17] which examines whether the variance in the inter-arrival times remains small. Specifically, the test traffic is separated into non-overlapping windows of size w packets. For each window i, the standard deviation σ_i of the inter-arrival times is computed. If the standard deviation of the pairwise differences between all pairs σ_i and σ_j for $i < j$ remains small, the traffic is considered highly regular and hence the presence of a covert channel is declared.
- **Entropy Tests.** Entropy-based detection test assumes that the presence of a covert timing channel has an effect on the entropy of a legitimate signal. To examine the entropy, the inter-arrival times are binned into Q bins which are used to calculate the empirical probability densities of patterns of size m in the sequence. A pattern is defined to be a sequence of bin numbers. The estimated entropy of patterns of size m is calculated using these probability densities. The regularity of the data can be measured from the entropy rate. Similarly a slightly different version of test using the conditional entropy estimates can be calculated. For more details see [19].

Other special purpose detection tests that are designed for specific covert timing channels include Cabuk et al.'s [17] ϵ-similarity for IP covert timing channels and Berk et al.'s [18] mean-max ratio to test for binary or multi-symbol covert timing channels. The mean-max ratio test assumes that the legitimate inter-packet delays follow a normal distribution which is often not true for real network traffic.

In Section VII, we show that none of the above tests can reliably detect the presence of covert communication.

VII. Experimental Results

The hardware-based covert channel spyware was implemented using the Wireless Open Access Research Platform (WARP) [4], which is a scalable, extensible programmable wireless platform that enables advanced wireless network prototyping. The WARP board contains reconfigurable hardware (FPGA) and can be connected by a UBS port to a PC running Windows. The encoder is implemented in hardware and its overhead when integrated within the WARP platform is reported for different encoder sizes in Section VII-A. The decoder is implemented in software using Matlab, since we do not study the spying receiver's architecture. We also do network level simulations where the communication among the different nodes introduces interfering network traffic. The goal is to evaluate the accuracy of decoding in presence of random interference as well as to study whether the covert flow survive traffic analysis. Our studies require two types of experimental setups: hardware setup and software simulation setup. The details of the hardware setup are as follows:

- The Vertix2P7 FPGA on the WARP platform is used for the encoder implementation. The spyware encoder is implemented in Verilog and integrated within the physical layer blocks on the FPGA.
- CSMA/CA MAC layer protocol is implemented on the PowerPC hardcore processor embedded in the WARP FPGA.
- The physical layer blocks with the exception of the analog front-end are all implemented on the WARP FPGA.
- A field size Q= 4 and GF(4) operations are used for encoding.
- Three different sizes of the encoding matrix (G) are implemented: 20×8, 40×16, and 80×32.

The details of the simulation setup are as follows:

- Aside from the spyware encoder and its intended receiver, there are K other nodes (possibly interfering) in the wireless network.
- The transmitting nodes send out the packets according to a Bernoulli process, with burstiness probability p.
- The spyware information is encoded in the timings of the packets, as discussed in Section IV-A, for a fixed encoder rate of $\frac{k}{n}\log_2 Q = 0.4$ bits/packet, and $\lambda = 0.225$ packets/sec.
- The collision management scheme in the CSMA/CA protocol introduces queuing of the packets - thus acting as a queuing timing channel - as discussed in Section III.
- The spyware decoder senses the transmission times of the wireless radio - which act as the departure process of the queue - and uses the iterative probabilistic decoder discussed in Section IV-B.

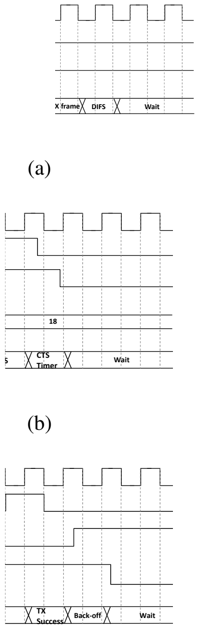
The timing diagrams in Figure 7 demonstrate the signal timings resulting from our WARP/FPGA implementation of the encoding matrix 20×8. Figure 7(a) contains the timing plot after initiating the CSMA/CA protocol by a trigger signal, denoted as *TX Ready*. In Figure 7(b) we show the initiation of the CTS timeout after sending an RTS packet. Figure 7(c) illustrates back-off after a successful frame transmission. The control state of the protocol is shown by the bottom waveform in all our timing diagrams. As expected, the spyware impact is covert and not detectable in timing signals. As we discussed in Section I, the performance of our spyware is evaluated with regards to three criteria: 1) transparency of the spyware circuitry, 2) robustness of the communication scheme to heterogeneous CSMA/CA effects, and 3) transparency of the covert communication scheme.

A. Transparency of the Spyware Circuitry

Here, we study the transparency of the spyware circuitry which is done by evaluating the cost of integrating the hardware encoder on the FPGA. The field size is set to 4 for the implementation. A template table implementing both addition and multiplication in $GF(4)$ is constructed and a C program is written to generate the encoder corresponding to any matrix

Original		Small (20 × 8)		Medium (40 × 16)		Large (80 × 32)	
		#	Overhead	#	Overhead	#	Overhead
Slices	24,518	24,531	0.05%	24,643	0.50%	24,993	1.94%
Gates	18,741,445	18,741,598	0.00%	18,742,966	0.01%	18,747,184	0.03%

TABLE I
THE OVERHEAD INTRODUCED BY THE ENCODER IMPLEMENTATION ON A WARP NODE FOR THREE ENCODING MATRIX SIZES. THE OVERHEAD IS REPORTED IN TERMS OF NUMBER OF ADDED SLICES/GATES TO THE FPGA.



(a)

(b)

(c)

Fig. 7. Timing diagram of WARP/FPGA implementation.

G of arbitrary size in Verilog. The encoder is inserted within the WARP platform physical blocks on the FPGA.

Table I shows the overhead introduced by the encoder implementation on a WARP node in terms of FPGA added slices and gates. The first column in Table I is labeled *original* and presents the original area of the wireless node components on WARP. The area is given in terms of the number of slices used on the FPGA and the estimated number of gates in the design. The next two columns show the area of the wireless node on the FPGA when we integrate an encoder with small matrix G of size 20×8. The percentage overhead in area in terms of FPGA slices is 0.05%, while the overhead in terms of the extra number of gates is negligible. The next four columns show the area and the percentage overheads when using a matrix G with size 40×16, and 80×32, respectively. Naturally, the overhead increases as the size of the matrix increases. Due to the very small area overhead, the integration of the encoder has no impact on the overall power of the WARP platform. The spyware introduced in this paper is so small compared to the rest of the circuitry that we are able to insert it on non-critical paths in our implementation. Thus, our encoder does not affect the overall delay of the digital circuitry implementing the radio because it does not affect the critical path of the digital design. The critical path timing is what determines the speed of the chip and is the timing information often provided within the chip specifications. Again we emphasize that the overhead is only measurable by the original designer, and the original unaltered files are never shared with any entity, including the foundry.

B. Robustness of the Communication Scheme to Heterogeneous CSMA/CA Effects

In this subsection, we study the robustness of the spyware communication in terms of the symbol error rate (SER) at the decoder. The SER is affected by the queuing effects introduced by the CSMA/CA due to varying network conditions. We generate multiple levels of interfering traffics on the network by simulating different number of nodes that communicate in the range of both the sender and receiver. It should be noted that the only purpose of the network traffic is to introduce interference that can affect the times of arriving packets carrying the spyware information. Table II shows the average SER at the decoder. These average SER values are shown for various numbers of interfering nodes, namely values of $K \in \{5, 10, 15\}$, as well as burstiness probability $p \in \{0.4, 0.6\}$ for three encoding matrix sizes $(k, n) \in \{(8, 20); (16, 40); (32, 80)\}$ corresponding to small, medium, and large on the table respectively.

Although the SER values are only on the order of 10^{-2}, we bring attention to the specific context of this application: a short code-length, externally triggered, and hardware implementation. Since this approach uses linear coset codes (which have extremely small undetected error probability [36]), an error event is essentially equivalent to the condition $H\hat{x} \neq s$. Thus, the spyware receiver can simply externally retrigger the transmission at a random time in the future. As it can be expected, Table II shows that as the length of the code (size of the encoding matrix) increases, the average symbol error rate at the decoder improves.

Figure 8 illustrates boxplots of the SER across multiple trials. The lower edge of the rectangular plot corresponds to $e_{.25}$, the 25th percentile; the upper edge pertains to $e_{.75}$, the $75th$ percentile. The 50th percentile, or median, appears as the horizontal line between the lower and upper bounds. The upper 'tail' of the box corresponds to the vertical line extending beyond the $75th$ percentile, which is of length $1.5(e_{.75} - e_{.25})$. All outliers extending beyond the 'tail' are explicitly drawn with the '$+$' symbol. We see from Figure 8 that the statistical structure of the SER is highly concentrated near 0, in such a way that cannot be evidenced by merely the mean SER. Specifically, the 25th percentile of the errors is 0. In terms of externally triggered retransmissions, this means that simple "majority-rules" decoding schemes will work particularly well with this approach. More specifically, while the average SER is of the order of 10^{-2}, around 25% of the time, the decoder receives symbols with zero error and thus a majority decoding rule after 2 or 3 transmissions will result in perfect decoding.

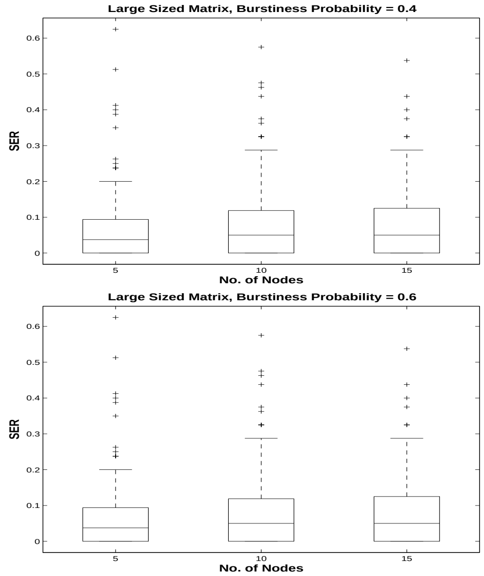


Fig. 8. Boxplots of the symbol errors across multiple trials, using the large encoding matrix with burstiness probabilities $p \in \{0.4, 0.6\}$ with 200 runs.

Number of nodes	Burstiness prob. = 0.4			Burstiness prob. = 0.6		
	Small	Medium	Large	Small	Medium	Large
5	.16	.08	.08	.13	.09	.05
10	.14	.10	.09	.14	.10	.07
15	.14	.09	.08	.13	.09	.08

TABLE II
AVERAGE SER OF THE SPYWARE DECODER VS. NUMBER OF CONGESTING NODES. THE STUDY IS DONE FOR DIFFERENT PROBABILITIES OF SENDING PACKETS OVER THE WIRELESS MEDIUM, AND THE MATRIX SIZES.

C. Transparency of the Covert Communication Scheme

In this subsection, we study the detectability of the covert information hidden in the network traffic. This is done by generating regular traffic and comparing it to spyware's covert traffic for various number of interfering nodes. The simulated traffic was generated and tested using Matlab on a 64-bit Six-Core AMD Opteron(tm) Processor 8431. To test the visibility of covert traffic, we compared two types of simulated traffic: (i) Regular traffic: simulated traffic generated using a Bernoulli distribution with burstiness probability p with no covert message embedded, and subjected to the CSMA/CA back off; (ii) Covert traffic: simulated traffic with a covert message embedded. This covert message was embedded using the scheme in section IV-A and the specifications in the beginning of section VII and in section VII-C. This traffic was then subjected to the CSMA/CA queuing effects for K number of interfering nodes. Inter-arrival times from the legitimate and covert traffic were used as inputs to three covert channel detection tests: Kolmogorov-Smirnov, regularity, and entropy tests of Section VI-B.

Number of packets	P_D		
	Small	Medium	Large
250	0.0357	0.2820	0.9022
500	0.0164	0.0280	0.3288
1000	0.0114	0.0195	0.0258
2000	0.0106	0.0122	0.0206

TABLE III
DETECTION (P_D) PROBABILITIES FOR USING A KOMOLGOROV-SMIRNOV TEST ON THE PACKET INTER-ARRIVAL TIMES (FALSE ALARM PROBABILITY FIXED AT 1%.)

Table III gives the results for the Kolmogorov-Smirnov test when a covert message was inserted in flows consisting of 10000 packets. In each of these flows, we randomly injected an external trigger activating covert transmission consisting of $n = 20$ (small), $n = 40$ (medium), or $n = 80$ (large) consecutive packets in every interval consisting of 250, 500, 1000 or 2000 packets. In all cases, the threshold for the hypothesis test was the deviation of the Komolgorov-Smirnov statistic beyond the 99% confidence interval, and the results are an average of over 10000 trials. When the covert message was inserted in every 1000 or 2000 packets, our approach was successfully undetectable – the detection probability, P_D and the false alarm probability, P_{FA} (fixed at 1%) are close to one another. When injecting the covert transmission within every 250 and 500 packets, the transmission becomes more visible, although the detection rates remain low except when $n = 80$ over 250 packets, in which case most of the covert transmissions are detected.

Number of packets	P_D		
	Small	Medium	Large
250	0.315	0.874	0.948
500	0.147	0.370	0.885
1000	0.058	0.108	0.228
2000	0.029	0.065	0.105

TABLE IV
DETECTION (P_D) PROBABILITIES FOR THE REGULARITY TEST (WINDOW SIZE $w = 100$ PACKETS AND FALSE ALARM PROBABILITY FIXED AT 1%).

Table IV illustrates the results for the regularity test with non-overlapping windows of size $w = 100$ packets over a collection of 250, 500, 1000 or 2000 packets. The results are averaged over a 1000 trials and the threshold was set beyond the 99% confidence interval. The number of possibly interfering nodes and the burstiness probability were chosen to be $K = 5$ and $p = 0.4$, respectively. Again, we randomly injected an external trigger activating covert transmission of $n = 20$ (small), $n = 40$ (medium), or $n = 80$ (large) consecutive packets. As mentioned earlier in Section VI-B, the regularity test of Cabuk et al. [17] examines whether the traffic is more regular than it is expected with legitimate traffic. Specifically, it tests whether the variance of inter-arrival times remains small. As seen in Table IV, the regularity test completely fails to detect the covert communication when the covert message was inserted in every 1000 or 2000.

In the following entropy detection tests, in accordance with recommendations of [19], we used bin sizes of $Q = 5$

and $Q = 65,536$, subsequently for the conditional entropy and the entropy tests. Tables V and VI shows the detection probabilities, P_D (P_{FA} fixed at 1%) for the two entropy-based detection tests on an infected traffic consisting of 10000 packets, where the covert transmission was inserted in every interval consisting of 250, 500, 1000 or 2000 packets. The external trigger activating covert transmission of $n = 20$ (small), $n = 40$ (medium), or $n = 80$ (large) consecutive packets was done at random positions in each interval. The threshold was set beyond the 99% confidence interval and the results are an average over 1000 trials. The number of possibly interfering nodes and the burstiness probability were chosen to be $K = 5$ and $p = 0.4$, subsequently.

Number	P_D		
of packets	Small	Medium	Large
250	0.206	0.319	0.419
500	0.144	0.245	0.441
1000	0.058	0.082	0.178
2000	0.061	0.064	0.090

TABLE V
DETECTION PROBABILITIES, P_D FOR ENTROPY TEST ON THE PACKET INTER-ARRIVAL TIMES IN PRESENCE OF RANDOM TRIGGER (FALSE ALARM PROBABILITY FIXED AT 1%.).

Number	P_D		
of packets	Small	Medium	Large
250	0.248	0.646	0.877
500	0.094	0.232	0.656
1000	0.043	0.050	0.148
2000	0.046	0.056	0.066

TABLE VI
DETECTION PROBABILITIES, P_D FOR CONDITIONAL ENTROPY TEST ON THE PACKET INTER-ARRIVAL TIMES IN PRESENCE OF RANDOM TRIGGER (FALSE ALARM PROBABILITY FIXED AT 1%.).

To see the effect of number of possibly interfering streams on the entropy-based, we tested the detection and false alarm probabilities when $n = 20$ (small), $n = 40$ (medium), or $n = 80$ (large) packets containing a covert transmission were sent. In this case, we did not embed the packets in a larger transmission sequence. Table VII illustrates the results for $K = 5, 10, 15$ interfering nodes still with a burstiness probability of $p = 0.4$ for the conditional entropy case. The results for entropy test were similar and hence omitted for sake of brevity. It can be seen that as number of interfering nodes increase the detection performance of the conditional entropy test decreases. One possible explanation is that because CSMA back off protocol engages more often to ensure collision avoidance, more packet timings are altered from their regular pattern, resulting in more randomness. This increase randomness in the streams seems to impair the detection test.

As clearly demonstrated by our simulations, none of the three detection tests can reliably detect presence of covert communication. In fact, often times the false alarm rates are comparable, if not higher, than the detection rates. However, note that the best transparency results are achieved when smaller number of covert packets are inserted more sporadically into the traffic, thus effectively reducing the transmission rate of the covert communication.

VIII. CONCLUSION AND FUTURE WORK

We have shown a new methodology for the design and implementation of spyware communication circuits that exploits the CSMA/CA MAC protocol properties to covertly leak the chip data to an intended spyware receiver. Our proposed scheme is robust to the collision avoidance techniques in the CSMA/CA protocol of the MAC layer. We interweave the encoder in the states and transitions of the wireless transmitter pre-synthesis, such that distinction and uncoupling of the spyware from the radio's physical layer is impossible. The spying act is initiated upon an occasional external or internal trigger. The proof-of-concept implementation is demonstrated on the widely used Wireless Open Access Research Platform (WARP). Three metrics were used for evaluation of the spyware performance: 1) efficiency of the encoder implementation, which was theoretically demonstrated; 2) robustness of the communication method to CSMA/CA effects, which was done by simulating the wireless architecture across a range of parameters; 3) difficulty of covert channel detection that was shown to be resilient against the known covert timing channels detection methodologies. While there exists a trade-off between accuracy, overhead and detectability, our experimental results show that our implemented spyware simultaneously meets the desired above metrics.

Some possible future directions include detection of the "trigger" time, or the time at which covert communication begins. One may think about implementing the spyware in other timing aspects of the wireless network such as rate adaptation or sleeping mode. It can potentially be done by the use of "change point detection" approaches [37] where they find the location in time where inter-arrival times drawn from a distribution F_0 switch to a distribution F_1.

However, if the circuit-level design criterion permits more sophisticated shaping schemes, even the change-point detection will fail. For instance, the encoder's shaping technique (3) of Section IV can be used to design provably undetectable covert communication schemes. At a high level, the idea is that the dithering scheme discussed in Section IV has two important properties:

- as we are using a random sparse graph linear coset code, it follows from [31] the x_i symbols are independent and identically distributed.
- the dithering technique in (3) allows each z_i to be i.i.d. with distribution given by $F_Z(z)$ used in the shaping.

It follows theoretically that any statistical test that models normal traffic as i.i.d. with inter-packet timings of distribution F_Z, will provably fail. More specifically, the inverse CDF, $F_Z^{-1}(\cdot)$, can be simply shaped to any desirable legitimate traffic model, as long as the inter-packet times are i.i.d.

REFERENCES

[1] V. Anantharam and S. Verdu, "Bits through queues," *IEEE Trans. on Information Theory*, vol. 42, no. 1, pp. 4–18, 1996.

Number of possibly interfering streams (K)	P_D			P_{FA}		
	Small	Medium	Large	Small	Medium	Large
5	.0095	.0130	0.0195	.0120	.0095	.0070
10	.0030	.0020	0.0120	.0030	.0020	0.020
15	.0003	.0020	0.0250	.0003	.0020	.0040

TABLE VII

DETECTION (P_D) AND FALSE ALARM (P_{FA}) PROBABILITIES FOR CONDITIONAL ENTROPY TEST VS. NUMBER OF POSSIBLY INTERFERING STREAMS (K).

[2] T. Coleman and N. Kiyavash, "Sparse graph codes and practical decoding algorithms for communicating over packet timings in networks," *Conference on Information Sciences and Systems*, pp. 447–452, 2008.

[3] S. Verdu, "The exponential distribution in information theory," *Problems of Information Transmission*, vol. 32, no. 1, pp. 86–95, 1996.

[4] "WARP: Wireless Open Access Research Platform," 2008, http://warp.rice.edu/trac/.

[5] "Defense Science Board (DSB) study on high performance microchip supply," 2005, http://www.acq.osd.mil/dsb/reports/2005-02-HPMS_Report_Final.pdf.

[6] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using ic fingerprinting," in *IEEE Symposium on Security and Privacy*, 2007, pp. 296–310.

[7] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *USENIX Security*, 2007, pp. 291–306.

[8] Y. Jin and Y. Makris, "Hardware trojans in wireless cryptographic ICs," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 26–35, 2010.

[9] S. Adee, "The hunt for the kill switch," May 2008, http://www.spectrum.ieee.org/may08/6171.

[10] S. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou., "Designing and implementing malicious hardware," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[11] B. Lampson, "A note on the confinement problem," *Communications of ACM*, vol. 16, 1973.

[12] A. Wagner and V. Anantharam, "Information theory of covert timing channels," in *NATO/ASI Workshop on Network Security and Intrusion Detection*, 2005.

[13] M. Kang and I. Moskowitz, "A pump for rapid, reliable, secure communication," in *ACM conference on Computer and communications security (CCS)*, 1993, pp. 119–129.

[14] V. Anantharam and S. Verdu, "Reflections on the 1998 information theory society paper award: Bits through queues," *IEEE Information Theory Society Newsletter*, pp. 100–102, December 1999.

[15] X. Luo, E. W. Chan, and R. K. C. Chang, "Covert communications in TCP burstiness," Hong Kong Polytechnic University, preprint, 2007.

[16] J. Giles and B. Hajek, "An information-theoretic and game-theoretic study of timing channels," *IEEE Trans. on Information Theory,*, vol. 48, no. 9, pp. 2455–2477, 2002.

[17] S. Cabuk, C. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *ACM conference on Computer and Communications Security (CCS)*, 2004, pp. 178—187.

[18] V. Berk, A. Giani, and G. Cybenko, "Detection of covert channel encoding in network packet delays," Dartmouth College, Dept of Computer Science Tech. Rep. TR2005536, 2005.

[19] S. Gianvecchio and H. Wang, "Detecting covert timing channels: an entropy-based approach," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 307–316.

[20] S. Kadloor, X. Gong, N. Kiyavash, T. Tezcan, and N. Borisov, "A low-cost side channel traffic analysis attack in packet networks," in *IEEE ICC 2010 - Communication and Information System Security Symposium*, Cape Town, South Africa, 2010.

[21] X. Gong, N. Kiyavash, and N. Borisov, "Fingerprinting websites using remote traffic analysis," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 684–686.

[22] S. Kadloor, X. Gong, N. Kiyavash, and P. Venkitasubramaniam, "Designing router scheduling policies: a privacy perspective," *Signal Processing, IEEE Transactions on*, vol. 60, no. 4, pp. 2001–2012, 2012.

[23] "IEEE 802.11 LAN/MAN Wireless LANS Standard," 2007, http://standards.ieee.org/getieee802/802.11.html.

[24] S. Wei, K. Li, F. Koushanfar, and M. Potkonjak, "Hardware trojan horse benchmark via optimal creation and placement of malicious circuitry," in *Design Automation Conference (DAC)*, 2012, pp. 90–95.

[25] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojans: Taxonomy and detection," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[26] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *IEEE Computers*, vol. 43, no. 10, pp. 39–46, 2010.

[27] F. Koushanfar and A. Mirhoseini, "A unified framework for multimodal submodular integrated circuits trojan detection," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 1, pp. 162 –174, March 2011.

[28] F. Koushanfar, "Provably secure active ic metering techniques for piracy avoidance and digital rights management," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 51–63, 2012.

[29] "Trusted computer system evaluation," U.S. Department of Defense, Tech. Rep. DOD 5200.28-STD, 1985.

[30] R. Sundaresan and S. Verdu, "Robust decoding for timing channels," *IEEE Trans. on Information Theory*, vol. 46, no. 2, pp. 405–419, 2000.

[31] R. G. Gallager, *Information Theory and Reliable Communication*. New York: John Wiley & Sons, 1968.

[32] A. Bennatan and D. Burshtein, "Design and analysis of nonbinary LDPC codes for arbitrary discrete-memoryless channels," *IEEE Trans. on Information Theory*, vol. 52, no. 2, pp. 549–583, 2006.

[33] C. Fragouloi, R. D. Wesel, D. Sommer, and G. Fettweis, "Turbo codes with nonuniform QAM constellations," in *IEEE International Conference on Communication*, 2001, pp. 70–73.

[34] F.-W. Sun and H. van Tilborg, "Approaching capacity by equiprobable signaling on the Gaussian channel," *IEEE Trans. on Information Theory*, vol. 39, no. 5, pp. 1714–1716, 1993.

[35] J. Ziv, "On universal quantization," *IEEE Trans. on Information Theory*, vol. 31, no. 3, pp. 344–347, 1985.

[36] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge University Press.

[37] A. Tartakovsky and V. Veeravalli, "Asymptotic analysis of Bayesian quickest change detection procedures," *IEEE International Symposium on Information Theory*, 2002.

Final Report of University of South Carolina Equipment Subcontract Grant

PI: Chin-Tser Huang

How this instrumentation has benefited the PI's research: In 2010, the PI used a previous AFRL grant (FA8750-09-2-0157, Project 1046148 / Award 49864) to purchase one server, four routers, and four interface cards and set up a preliminary testbed network. However, that testbed only had a single-line routing path, which did not reflect the complex network topology and routing used in the Internet. With the support of this grant, we were able to purchase four Cisco 2911 routers and two Cisco 2921 routers, along with the necessary peripherals. The procured instrumentation increased the total number of our routers to 10 and has allowed us to set up a more complex network topology and conduct experiments that involve overlapping routing paths and crossover routing paths. The collected traffic data will also be experimented and analyzed using the testbed.

How this instrumentation will continue to benefit the PI's research and collaboration with AFRL: The PI has an ongoing NSF project "TC: Small: Dynamic Early Filtering of Botnet Garbage Traffic" (Award CNS-0916857) which aims to develop a comprehensive and sustainable architecture to coordinate the routers in the Internet to achieve early filtering of unwanted botnet packets in Internet traffic. The instrumentation procured from this grant is significant and integral for the PI to set up a sensible testbed that allows the PI and his students to conduct experiments resembling the real traffic and routing in the Internet, such that the results could bring convincing contribution to the collective efforts in countering and mitigating botnet attacks.

The PI has participated in the Summer Faculty Fellow Program (SFFP) funded by AFOSR and the Visiting Faculty Research Program (VFRP) funded by AFRL since summer 2008, and has been collaborating with the CyberBAT Lab led by Dr. Keesook Han. The collaboration is still ongoing through the EPA between AFRL and USC. The instrumentation will allow the PI to conduct experiments relevant to the collaboration with AFRL.

Publications benefited from the instrumentation:

[1] C.-T. Huang, K. Han, H. Carroll, J. Perretta. Improving Cloud Performance with Router-based Filtering. In the book *High Performance Semantic Cloud Auditing*, to be published by Springer, 2013.

[2] C.-T. Huang, K. J. Han, J. Perretta. Automatic Selection of Routers for Placing Early Filters of Malicious Traffic. *Proceedings of 54th Annual IEEE Global Telecommunications Conference* (GLOBECOM 2011), December 2011.

Improving Cloud Performance with Router-based Filtering

Chin-Tser Huang, Keesook J. Han, Heath Carroll, and James Perretta

Abstract We introduce a router-based filtering technology aimed to enhance the security and performance of cloud computing. When this technology is integrated with cloud auditing methods, it can make use of the cloud auditing information to detect malicious intrusion and traffic anomalies, and define appropriate filtering rules that can be exchanged between routers in the network, such that it can filter malicious traffic early and reroute the excessive legitimate requests to other suitable replicated servers. We first give an overview of the specification and generation of filtering rules used by routers. Then we present a theoretical model to find the best locations for hardware routers in a network to block malicious traffic, and introduce how to integrate this theoretical model with cloud auditing techniques. Finally, we will discuss the experiments conducted to evaluate the reliable theoretical model and the dynamic programming based algorithm for router-based filtering and rerouting in cloud computing. The results validate the effectiveness and benefits of our theoretical model and algorithm.

1 Introduction

Cloud computing has attracted plenty of attentions and interests because it realizes the long-desired goal of on-demand network access to a scalable shared pool

Chin-Tser Huang
University of South Carolina, Columbia, South Carolina, USA, e-mail: huangct@cec.sc.edu

Keesook J. Han
Air Force Research Laboratory, Rome, New York, USA, e-mail: Keesook.Han@rl.af.mil

Heath Carroll
University of South Carolina, Columbia, South Carolina, USA, e-mail: carrollh@cec.sc.edu

James Perretta
Air Force Research Laboratory, Rome, New York, USA, e-mail: James.Perretta@rl.af.mil

of computing resources, such as servers, storage, applications, platforms, and networks. However, along with cloud computing also come concerns about security issues, such as authentication, integrity, privacy, and availability. In this chapter, our main focus is to address the availability issues.

Availability in cloud computing can be considered from two aspects: communication channel availability and cloud resource availability. Communication channel availability is gaining more significance as more and more services are migrating to the cloud. Since clients entrust cloud servers to meet their storage and computation needs, it is important that clients can get the needed service when they need it. In particular, many real-time cloud applications have emerged in these past few years, such as teleconference, remote surgery, massively multi-player online games, and so on. Real-time delivery of the service is paramount for these time-sensitive applications [1]. If the cloud in question is a public cloud, the clients and the servers are not in the same domain; in this case the Internet is required to deliver the service. However, the reliability of the Internet is often impacted by rampant DoS and DDoS attacks. If the unwanted packets from these attacks are allowed to traverse the routers and links in transit to their destinations, they can consume huge amount of router bandwidth and link bandwidth, and disrupt the timely delivery of legitimate packets. On the other hand, cloud resource availability is no less important than communication channel availability. Cloud service providers try their best effort to analyze and predict the pattern of client needs, and allocate the cloud resources accordingly. However, the level of service requests from clients is always dynamic. To ensure that cloud resources can continue to meet client needs, the cloud service provider must be able to quickly readjust the allocation of cloud resources when facing excessive level of service requests, or reroute service requests to replicated servers that provide backup services.

Our goal in this book chapter is to introduce a router-based filtering technology aimed to enhance the availability and performance of cloud computing. When this technology is integrated with cloud auditing methods, it can make use of the cloud auditing information to detect malicious intrusion and traffic anomalies, and define appropriate filtering rules that can be exchanged between routers in the network, such that it can filter malicious traffic early and reroute the excessive legitimate requests to other suitable replicated servers.

This chapter is organized as follows. We first give an overview of the specification and generation of filtering rules used by routers. Then we will present a theoretical model to find the best locations for hardware routers in a network to block malicious traffic, and introduce how to integrate this theoretical model with cloud auditing techniques. Finally, we will discuss the experiments conducted to evaluate the reliable theoretical model and the dynamic programming based algorithm for router-based filtering and rerouting in cloud computing. Our experimental results validate the effectiveness and benefits of our theoretical model and algorithm.

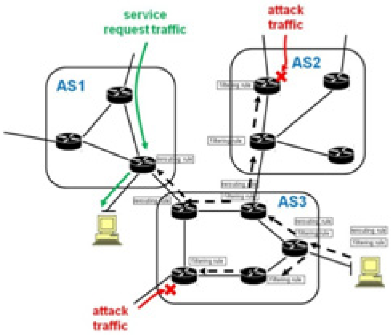


Fig. 1 Router-based filtering and redirection.

2 Router-based Filtering and BGP Flow-Spec Rules

Routers are originally designed to forward packets toward their specified destination according to the routing table, which contains entries specifying the routing information they exchange with adjacent routers. However, it is also suitable to use routers for access control and filtering [4] since packets need to pass through routers on their way to the destination. Compared to using traditional firewalls installed at the end host to filter malicious unwanted traffic, using routers for filtering has the potential to filter the malicious unwanted traffic at a location closer to the source, therefore saving the bandwidth that would otherwise be consumed by the malicious unwanted traffic. An additional benefit is that using routers for filtering allows for the realization of multiple instantiations of dynamic filtering rules because routers frequently communicate with neighboring peer routers to advertise new routes or route changes and update their routing tables. Routers can disseminate filtering rules along with routing information update messages. If their routing tables are appropriately updated, routers can be instructed to discard malicious packets immediately, limit the rate of malicious packets, redirect legitimate packets to a different legitimate server, or forward them to a security center for further analysis and diagnosis. Fig. 1 illustrates the operation of this router-based filtering and redirection technology.

In order to coordinate the routers in the Internet to perform the task of early filtering of malicious unwanted traffic, we need a means for routers to exchange filtering rules with adjacent routers. The flow specification (Flow-Spec) rules defined for the Border Gateway Protocol (BGP) [13] in [9] provide such a way. If filtering rules are defined as Flow-Spec rules, they can be passed using BGP UPDATE messages. Since the purpose is to filter the malicious unwanted traffic closer to the source, and BGP is the de facto inter-domain routing protocol, BGP UPDATE message is the appropriate message to use to forward the filtering rules upstream toward malicious traffic sources.

BGP is used to exchange routing information across different domains in the Internet and is the protocol used between ISPs. A domain, or autonomous system (AS), is a network or group of networks under a common administration and with common routing policies. After two peer BGP routers establish a connection, they will use an OPEN message to set up initial parameters, and exchange their routing table. The two BGP routers will send periodical KEEPALIVE messages if there is no change in the routes, and use UPDATE message to convey any route change information to the other router. BGP UPDATE messages contain a field called Network Layer Reachability Information (NLRI). The Flow-Spec rule is defined as one new type of NLRI, so that it can be carried by BGP Update messages. Formally, a Flow-Spec rule is an n-tuple consisting of several matching criteria that can be applied to IP traffic. The matching criteria of a Flow-Spec are specified by encoding various characteristics of the offending traffic, including destination prefix, source prefix, IP protocol, port, destination port, source port, ICMP type, ICMP code, TCP flags, Packet length, DSCP, and Fragment. If a given packet satisfies the conjunction of all the specified criteria of a defined Flow-Spec, it is considered a match to the Flow-Spec and should be filtered.

A Flow-Spec received from an external domain needs to be validated against unicast routing before being accepted and installed. The underlying principle of validation is to check that a BGP router from a neighboring AS that advertises a best unicast route for a destination can only advertise a Flow-Spec rule which contains a more or equally specific destination prefix. Moreover, neighboring domains need to establish trust with each other that they will only advertise valid reachability information and will not advertise an NLRI on behalf of a prefix for which it does not provide service; otherwise it may cause a denial-of-service attack on that given prefix. More details about the validation of the Flow-Specs and the establishment of trust between neighboring domains are discussed in [9].

As of now, BGP Flow-Spec has received support from multiple major vendors such as Juniper [10] and Arbor. Cisco has also announced plans to incorporate BGP Flow-Spec into their routers [5]. Although direct Flow-Spec support is not yet available in Cisco routers, it is possible to use Ciscos existing Access Control List (ACL) to emulate a subset of Flow-Specs features. In our own research projects, we have successfully implemented BGP Flow-Spec rules on a routing software suite called Quagga [12], and have also developed Tcl scripts which allow Cisco routers to exchange ACL rules with other routers and install received rules.

In the following subsections, we continue to introduce the filtering actions associated with Flow-Spec rules and the factors associated with the router-based early filtering approach.

2.1 Filtering Actions

A router continues to compare the received traffic with the Flow-Spec rules in its rule base according to their order in the rule base, until a matching rule is found or the end of the rule base is reached with no matching rule being found. When the router finds a matching Flow-Spec rule, i.e. the traffic matches all the criteria of the rule, the router will apply the action specified in the rule on the traffic. The BGP Flow-Spec standard [9] specifies a minimum set of typical filtering actions beyond the default action in which the traffic matching the Flow-Spec rule in question is accepted. Four types of BGP extended community values are defined to standardize the minimum set of filtering actions, which are described as follows.

The first type is called *traffic-rate extended community*, which is often used for policing applications. The encoding is 6 bytes. The first 2 bytes stand for an AS number, while the remaining 4 bytes carry rate information in IEEE floating point format, with the units expressed in bytes per second. The traffic in question is totally filtered if the rate is specified as 0.

The second type is called *traffic-action extended community*, which consists of 6 bytes but currently only the 2 least significant bits of the 6th byte (from left to right) are defined. When the Terminal Action bit (bit 47) is set, subsequent filtering rules will be applied by the filtering engine if the current rule is not a match. If this bit is not set, subsequent filtering rules will not be applied no matter the current rule is a match or not. When the Sampling bit (bit 46) is set, traffic sampling and logging for this flow is enabled.

The third type is called *redirect extended community*, which allows the traffic to be redirected to a VRF routing instance. Its 6-byte value encodes the route target.

The fourth type is called *traffic marking extended community*, which instructs a system to modify the DSCP bits of a transiting IP packet to the corresponding value. This extended community value is encoded as a sequence of 5 zero bytes followed by the DSCP value encoded in the 6 least significant bits of the 6th byte.

Although the above extended community values have been defined, the standard specification leaves it open for each unique implementation to use arbitrary extended community values for various additional filtering actions, as heterogeneous networks and routing devices make it difficult to standardize on all possible filtering actions.

2.2 Factors of Router-based Filtering

It is expected that the router-based early filtering approach can filter the malicious unwanted traffic early and so can benefit the legitimate traffic with reduced latency and increased throughput. However, the benefits are not always guaranteed along with the deployment. This is mainly due to the cost of filtering: a routers action of filtering packets against a rule will incur extra packet processing overhead, and all the packets, no matter unwanted packets or legitimate packets, need to be matched against the installed rules. In this subsection, we discuss the factors that need to be considered to ensure the effectiveness and benefits of the router-based early filtering approach.

1. **Degree of Rule Dissemination**: Many unwanted packets can be attributed to DDoS attacks launched by botnets, in which the sources are highly distributed. In this case, if a corresponding filtering rule is disseminated to more routers, then there is a better chance to filter more unwanted traffic at an earlier time. We regard the number of hops for which a rule is disseminated as its degree of dissemination. Depending on the type of attack, the rule can be disseminated at each hop to only one specific upstream router (if the offending source is already known) or to every neighboring router except the one from which the rule is received. However, higher degree of rule dissemination also increases the overall filtering overhead.
2. **Filtering Overhead**: Although filtering unwanted traffic early can save bandwidth for more efficient network utilization by legitimate traffic, this benefit does not come for free. The action of filtering itself will consume the computation resource of a router, which will slow down the routers packet processing to some degree. Therefore, to ensure that there is a considerable net gain, it is necessary to consider the tradeoffs between the bandwidth saved from the filtered unwanted traffic and the overhead resulting from early filtering. Assume that the size of the attacking botnet (i.e. the number of participating bots in the botnet) is N, and each participating bot contributes traffic of average bandwidth u to the victim. If the filtering rule is disseminated outbound for h hops in a breadth-first manner, the overall bandwidth we can save by filtering the attack traffic is roughly $h \cdot N \cdot u$ (assuming the sources are at least h hops away from the victim). On the other hand, the use of filtering rules will lead to a slowdown in packet processing. Assume that the rule corresponding to the attack is installed at i routers, and the overhead of filtering against one rule will result in an average reduction of th in throughput. Thus, the cost for achieving the above saving is estimated at $i \cdot th$. The goal is that the saving is larger than the cost, which is very likely considering the large size of botnets and the ever growing processing speed and capabilities of routers.
3. **Size of Rule Cache and Replacement of an Old Rule**: It is desirable to limit the filtering overhead on each router to an acceptable range. Besides limiting the degree of rule dissemination, we can also specify the maximum number of rules each router installs. The installed rules can be maintained in a rule cache. When a new filtering rule is received and the rule cache is full, the router has to make a decision about whether to drop the newly received rule or to replace an old rule with the newly received rule.
4. **Revocation of Obsolete Rules**: A filtering rule will not stay valid forever because most offending unwanted traffic will eventually stop. It is desirable to revoke obsolete rules in order to save filtering overhead. We can revoke obsolete rules with two methods. First, when the victim of the unwanted traffic detects that the attack has stopped, it can send out a revocation rule to notify the routers to revoke the corresponding obsolete rule. The revocation rule must be authenticated to prevent an attacker from spoofing a request to revoke an effective filtering rule. Second, it is also possible to include a lifetime or expiration time in the filtering rule, so that the routers know when to revoke a rule.

5. **Location of Legitimate Sources**: For a router, installing a filtering rule spells for more processing overhead. Therefore, the dissemination of filtering rule should not continue without a bound. When considering how far should a filtering rule be disseminated, one important observation is that at any given time a host only communicates with a certain number of legitimate hosts. Assume that the victim host in question is communicating with n other legitimate hosts, and the number of hops from the victim to each of these hosts is $h_1, h_2, \ldots, h_n$. We can call the largest value among $h_1, h_2, \ldots, h_n$ as the communication radius for the victim. If the filtering rule is disseminated beyond the victims radius of communication, the legitimate sources will not get additional benefit, because the links over which the bandwidth is saved due to the extra dissemination are not on the paths from the legitimate sources to the victim. If a victim knows that it will only communicate with legitimate hosts within communication radius r, then the victim can include r in the filtering rule it sends out, so that the routers will not to disseminate the rule beyond r hops from the victim.

3 Selecting Appropriate Routers to Install Filtering Rules

One important problem that remains to be addressed is how to determine which set of routers in the network should be selected to install a given filtering rule of malicious traffic in order to maximize the benefit of early filtering. There are at least two reasons which make this problem important. First, in our previous work [8] we had shown that more router bandwidth can be saved by forwarding filtering rules to the routers closer to the attack sources. This is straightforward to realize if all the malicious traffic converges on the same single path, because moving away from the victim on the path means moving toward attack sources. However, in todays Internet the attacks and threats often originate from very distributed sources, which largely complicate this problem. Second, for consideration of router efficiency it is not possible to install unlimited number of filtering rules in a router because the overhead of matching every received packet against every filtering rule will be too large. To meet this constraint, it is desirable to install a filtering rule only in a chosen subset of routers instead of in all routers in the network.

In this section, we formally formulate the problem of automatic selection of routers for filter placement, and present an analytical solution to this problem which was also introduced in [7]. The proposed algorithm is based on dynamic programming. We start with the simple case of tree topology in which there is only one path from each node to the root, and then generalize it to cyclic topology. We also discuss how to apply our early filtering approach and the QoS based routing with traffic distribution in the networks.

We introduce a dynamic programming based algorithm to solve this problem of router selection. In our network model, we assume that the routing paths from the distributed attackers to the target server (the victim) form an acyclic tree topology, in which there is only one path from each attacker to the victim. (We begin our discussion with the tree topology for the sake of simplicity. In a later section, we will discuss an extension of our model and algorithm to cover cyclic topologies.) The root of the tree is the victim, and other nodes on the tree represent intermediate routers. Therefore, the network can be modeled as a tree $T = (V, E)$, where V is a set of routers and E is a set of links connecting adjacent routers in V. There may be a number of hosts, legitimate or malicious, connected to each router. Malicious hosts can send malicious traffic toward the victim. In our network model, our concern is focused on the routers, not the hosts, because all the malicious traffic flows are forwarded by the routers toward the victim. Without loss of generality, we consider one routing path at a time.

An example of our network model is shown in Fig. 2. In this example, if router R_3 has malicious traffic destined to the victim R_0, then R_3 will forward the malicious traffic to R_2, which is the next router on the path from R_3 to R_0. Next, R_2 will forward the malicious traffic received from R_3, along with the malicious traffic received from malicious hosts connected to R_2 (if any), to router R_1. Finally, R_1 will forward the malicious traffic received from R_2, along with the malicious traffic received from malicious hosts connected to R_1 (if any), to R_0. The goal of our approach is to install filtering rules at a set of appropriate routers in the network, such that the malicious traffic flows are blocked early, soon after their packets enter the network. This approach has at least two significant benefits. First, it can save bandwidth of the routers and links from being occupied by malicious traffic, and make the saved bandwidth available for the use of legitimate traffic. Second, it can confine other possible damages that could be caused by the malicious traffic (e.g., slowing down the hosts or routers, modifying or deleting files in the hosts or routers) to local regions. Fig. 3 illustrates the case in which some malicious traffic flows are blocked at routers where the corresponding filtering rule is installed (represented by a gray circle). In this example, since router R_3 installs the corresponding filtering rule, R_3 will block the malicious traffic it receives. As such, there will be no malicious traffic flowing from router R_3 to router R_2. Router R_2 does not installs the corresponding filtering rule, so R_2 will forward the malicious traffic it receives toward router R_1. Router R_1 installs the corresponding filtering rule, so R_1 will block the malicious traffic it receives, and there will be no malicious traffic flowing from router R_1 to the victim R_0.

To formulate the problem, we have to first define the saving and cost resulting from filtering a malicious traffic flow at a router u. It is apparent to see that there is a miss penalty if one specific malicious traffic flow is not filtered at a given router u. The miss penalty includes at least the following two components: (1) the bandwidth (of the link to the next router on the routing path to the victim) that will be occupied by the malicious flow if not filtered, (2) the cost of the threat or damage caused by the malicious flow if not filtered. The first component is apparently a numeric value which could be normalized according to the capability of the router and the capacity of the link, while the second component needs to be quantified using some predefined metrics. Let $m(u, f_i)$ be the miss penalty of allowing one unit of malicious traffic flow f_i (e.g., one individual packet of the malicious flow, or a set of consecutive packets that can cause damage when allowed to pass) at router u. We

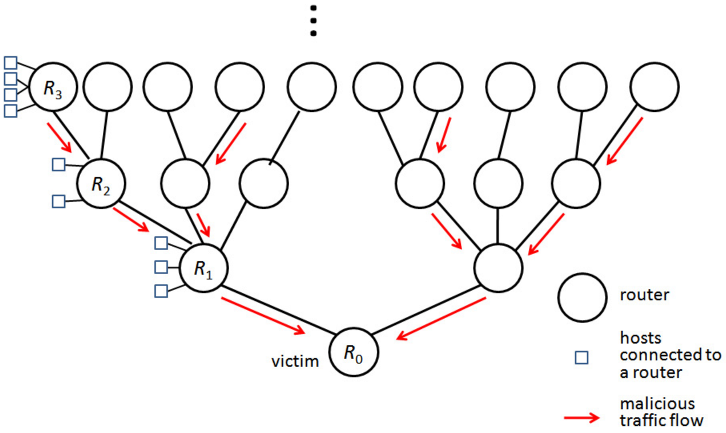


Fig. 2 The network model.

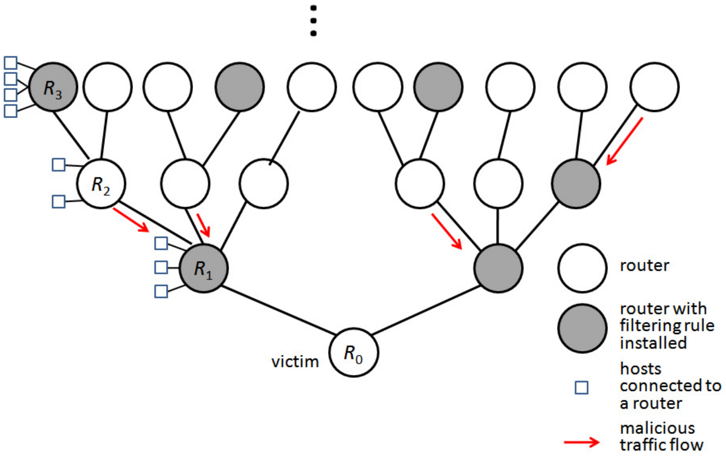

R_3
R_2
R_1
R_0
victim
router
router with filtering rule installed
hosts connected to a router
malicious traffic flow

Fig. 3 Malicious traffic flows are blocked at some routers where the corresponding filtering rule is installed.

can define $m(u, f_i)$ as

$$m(u, f_i) = \sum\nolimits_{(u_1,u_2) \in PATH(u,u')} c(u_1, u_2, f_i)$$

where u' is the nearest node on the path from u to the root that installs the filtering rule on f_i, and u_1, u_2 are adjacent nodes on the path from u to u' and $(u_1, u_2) \in E$, and $c(u_1, u_2, f_i)$ is a function that will output the penalty of not filtering one unit of f_i on the link from u_1 to u_2.

Next, we analyze the tradeoff between the saving and cost when early filtering is exerted. The bandwidth saved at a router due to filtering is related to the packet rate of the malicious traffic flow arriving at the router. If the rate of a malicious traffic flow is higher, then the bandwidth saved by filtering this flow is higher. On the other hand, installing a filtering rule at a router imposes processing overhead on the router because the router will have to match the filtering rule against every received packet to determine whether the packet should be allowed or blocked. (Even if a received packet is legitimate, it will be matched against every filtering rule so that it can be determined to be legitimate.) This overhead will negatively impact the throughput at the router. Let $r(u, f_i)$ be the rate of malicious flow f_i observed at router u, and $o(u, f_i)$ be the amount of reduced throughput due to the overhead of installing the filtering rule on f_i at router u. Then the saving of filtering f_i at router u can be derived as

$$r(u, f_i) \cdot m(u, f_i) - o(u, f_i)$$

In addition, there is one possible source of performance loss in the throughput. We note that due to the overhead of filtering there should be an upper bound on the number of installed filtering rules at each router, because if a router installs unlimited number of filtering rules then it may have to spend too much of its computation resources on filtering, thus leading to long delay and low throughput. If a router wants to install a new rule but the number of installed filtering rules at the router has reached the upper bound, then one existing rule should be removed in order to make space for installing this new rule. We assume that there is a heuristic algorithm available for a router to select the rule that generates the least saving to replace. In particular, let $l(u, f_i)$ be the performance loss in the throughput due to replacing the least-saving rule at router u, then $l(u, f_i)$ can be derived as

$$l(u, f_i) = min_{every\ f_j\ currently\ filtered\ at\ u}\{r(u, f_j) \cdot m(u, f_j)\}$$

It is clear that the rule selected to be replaced also imposes processing overhead on router u, whose value is the same as $o(u, f_i)$ because this new rule also needs to be matched against every received packet. Therefore, the net saving of filtering f_i at router u can be derived as

$$r(u, f_i) \cdot m(u, f_i) - o(u, f_i) - (l(u, f_i) - o(u, f_i)) = r(u, f_i) \cdot m(u, f_i) - l(u, f_i)$$

Now we can formulate the problem. For simplicity reason without loss of generality, we limit our consideration to only one routing path in the network and one type of malicious traffic. (Note that other routing paths in the network and other types of malicious traffic can be formulated in the same way.) Suppose one routing path of the malicious traffic contains n routers on the path, R_1 through R_n, and R_0 is the victim of the malicious traffic. The filtering rule corresponding to the given malicious traffic can be installed in up to k routers, where $0 \leq k \leq n$, on the routing path. This leads to an optimization problem on figuring out which routers (up to k) should install the filtering rule in order to maximize the saving.

Let r_i be the rate of the malicious traffic observed at router R_i, and m_i be the miss penalty of the malicious traffic at router R_i. Since every intermediate router may possibly be connected to malicious hosts that will inject more malicious traffic toward the victim R_0, we can get $r_1 \geq r_2 \geq \ldots \geq r_n$, which means a router closer to R_0 will observe higher rate of the malicious traffic. Assume $x \leq k$ intermediate routers R_{v_1}, R_{v_2}, $\ldots$, R_{v_x}, where v_1, v_2, $\ldots$, v_x are integers and $1 \leq v_1 \leq v_2 \leq \ldots \leq v_x \leq k$, are chosen to install the filtering rule. Since R_{v_i+1} would have filtered rate r_{v_i+1} of the malicious traffic, so the rate of malicious traffic to be observed and filtered by R_{v_i} is only $r_{v_i} - r_{v_i+1}$. Also, let l_i be the performance loss due to replacing an existing filtering rule at router R_i. Thus the overall saving achieved by filtering at the aforementioned x routers R_{v_1}, R_{v_2}, $\ldots$, R_{v_x} is given by the following summation:

$$\sum_{i=1}^{x}((r_{v_i} - r_{v_i+1}) \cdot m_{v_i} - l_{v_i})$$

Therefore, the problem can be formally defined as follows.

Definition 1. The filter placement *k*-optimization problem: Given R_0, R_1, R_2, $\ldots$, R_n, which are nodes on a tree $T = (V,E)$ and R_i is the parent of R_{i+1} on the tree T, and associated parameters r_1, r_2, $\ldots$, r_n, r_{n+1}, where $r_1 \geq r_2 \geq \ldots \geq r_n \geq r_{n+1} = 0$, m_1, m_2, $\ldots$, m_n, l_1, l_2, $\ldots$, l_n, and k be an integer where $0 \leq k \leq n$. Compute x, where $0 \leq x \leq k$, and a set of x integers $v_1, v_2, \ldots, v_x$ where $1 \leq v_1 \leq v_2 \leq \ldots \leq v_x \leq k$ (and $v_0 = 0$), such that the following objective function is maximized:

$$\Delta SAV(R_n, k : v_1, v_2, \ldots, v_x) = \sum_{i=1}^{x}((r_{v_i} \cdot \sum_{j=v_{i-1}+1}^{v_i} m_j) - l_{v_i})$$

In its general form, $\Delta SAV(R_{ind}, k)$ represents the net saving of the case when up to k of routers R_1 through R_{ind} can install filtering. Also, define $\Delta SAV(R_{ind}, 0) = 0$.

We propose to use a dynamic programming based algorithm to solve this k-optimization problem. Dynamic programming is a well-known method which solves complex problems by breaking it into smaller overlapping subproblems. To achieve this, a recurrence is needed to link the original problem with its subproblems.

The recurrence relation and initial condition required by the dynamic programming method are given as follows:

$$\Delta SAV(R_{ind}, k) = \begin{cases} max(max_{1 \leq i \leq ind}(\Delta SAV(R_i, k-1) + (r_{ind} \cdot \sum_{j=v_x+1}^{ind} m_j) - l_{ind}), \\ \quad \Delta SAV(R_{ind-1}, k)), & if\ ind \geq k; \\ \Delta SAV(R_{ind}, ind), & if\ ind < k \end{cases}$$

where the v_x in the first case is the last router (the one with largest index) found in the solution to $\Delta SAV(R_i, k-1)$.

3.1 Resolving Cycles in Network Topology

In the previous discussion, it is assumed that the network topology is tree-shaped for the sake of simplicity. However, in reality a network topology often contains one or more redundant paths between two routers, for several reasons such as reliability and load balancing. The undirected edges representing these paths form cycles in the topology.

There is one incompatibility between cyclic topology and our algorithm. When the malicious traffic reaches one router on the cycle, the malicious traffic may split into two portions, each following one different path toward the victim. This violates our basic assumption that a router closer to the root of the tree (in this case the victim) will observe higher rate of the malicious traffic due to convergence.

Our algorithm can be extended with the following operation to resolve the cycles in the topology. If a cycle is found in the topology, we can find the router at the split point where the malicious traffic splits into two paths, and install the filtering rule in that router. Therefore, no malicious traffic will flow from the router at the split point to the next router on each of the two paths. In this case, we can temporarily remove the two edges connecting the router at the split point and the next router on each of the two paths, and divide the topology into two subgraphs: one subgraph is rooted at the victim and the other is rooted at the router at the split point. With this operation, the assumption that a router closer to the root will observe higher rate of the malicious traffic is again satisfied. This operation can be applied multiple times until all the cycles in the topology are resolved. Then, our algorithm can be applied on each subgraph separately.

We can use the example topology in Fig. 4 to illustrate this extension. Suppose R_0 is the victim and the sequence (R_0, R_3, R_4, R_5, R_2, R_1, R_0) forms a cycle. When the malicious traffic reaches R_5, it will split into two paths $R_5R_4R_3R_0$ and $R_5R_2R_1R_0$. If the filtering rule is installed at R_5, then there will be no malicious traffic flowing from R_5 to R_4 and from R_5 to R_2. Therefore, we can divide the topology into two subgraphs, one rooted at R_0 and the other one rooted at R_5, and each subgraph satisfies the basic assumption.

3.2 Integrating Router-based Early Filtering and Cloud Auditing with QoS Routing Algorithms

It has been shown in [8] that our router-based early filtering approach can reduce the average transmission latency and increase the average throughput of legitimate traffic. The dynamic programming based filter placement algorithm presented above can find an appropriate set of routers automatically to install the filtering rule to maximize the benefit of early filtering. These improved values of transmission latency and average throughput can be used as feedback to QoS routing algorithms. QoS routing is a well studied topic and was proposed to provide guaranteed services by

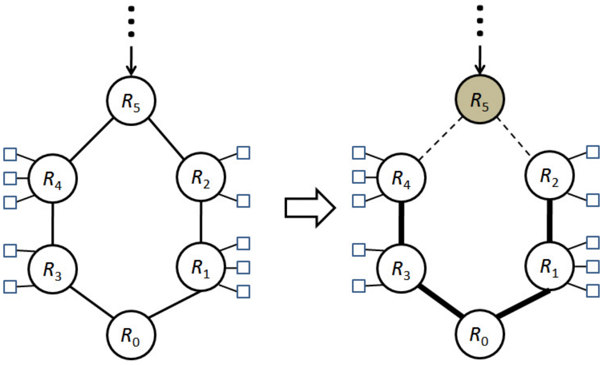


Fig. 4 Configuration of testbed network.

finding a path satisfying the QoS requirements of the traffic. There are existing QoS routing algorithms based on dynamic programming, graph theory, and probability, such as [3], [6], and [14].

In general, QoS routing algorithms depend on the link state information, such as available bandwidth, transmission latency, loss probability, and cost function, to make routing decisions that satisfy QoS requirements of specific traffic. Since the network environment changes as time goes by, the link state information which was given as the initial input to the QoS routing algorithm may not persist.

The router-based early filtering approach and cloud auditing techniques can be combined to improve the performance of QoS routing algorithms. Cloud auditing techniques are usually used to monitor the service level and the usage of cloud resources, and analyze the interaction between cloud servers and clients. If the early filtering approach is applied, the shorter transmission latency and higher average throughput resulting from the saved bandwidth will be observed and recorded by cloud auditing tools. When the improved values of transmission latency and average throughput are provided to QoS algorithms, they can make more precise routing decisions that utilize the benefits generated by the early filtering approach.

4 Experimental Results

We design a set of experiments to evaluate the performance and benefits of our approach introduced in Section 3. The steps we follow to conduct the experiments are summarized as follows:

Step 1. Set up a testbed network which is composed of a single path of routers, one host (playing the role of the distributed attacker) connected to each router, and the destination (the victim).

Step 2. Define the parameters for the experiment, including r_i, m_i, and l_i at router R_i.

Step 3. Inject the target attacking traffic and the secondary attacking traffic whose volume meet the defined parameters.

Step 4. By default, install in each router the filtering rule corresponding to the secondary attacking traffic that directly comes into each router instead of being forwarded by another router.

Step 5. Do the following for each possible combination of routers: for each router in the chosen combination, replace the filtering rule on the secondary attacking traffic with the filtering rule on the target attacking traffic; generate the legitimate traffic toward the destination and measure the throughput of the legitimate traffic.

Next, we describe each step of our simulation in detail. In Step 1, with our currently available equipment, we set up a testbed as illustrated in Fig. 5. The configuration of our testbed is composed of a single path of 4 Cisco 2911 routers, with a host connected to each router to simulate distributed attackers. The attackers are from the addresses 192.168.1.2, 192.168.2.2, 192.168.3.2, and 192.168.4.2 as shown in the figure. All the malicious traffic flows are destined to the victim host located at the address 192.168.5.2. To measure the performance of our algorithm, we also connect a host with the address 192.168.6.2 to the last router to send legitimate traffic to the victim host.

In Step 2, we define the parameters for the experiment. Since there are 4 routers on the path in the testbed, we call the 4 routers R_4, R_3, R_2, R_1 respectively, while R_0 represents the destination. We define the relevant parameters r_4, r_3, r_2, r_1, m_4, m_3, m_2, m_1, l_4, l_3, l_2, l_1, for the simulation example as shown in Fig. 6. Note that the value of each r_i must be at least equal to the value of $r_i + 1$; this is because router R_i may receive the attacking traffic from $R_i + 1$ and also from the attacker directly connected to R_i itself. Also note that we normalize the value of all m_i to 1 in this particular simulation because all the attacking traffic is ping packets and all the routers used in our testbed are of the same model.

In Step 3, we need to inject the target attacking traffic and the secondary attacking traffic which meet the parameters defined in Step 2. The target attacking traffic simulates the traffic which corresponds to the new filtering rule, therefore the volume of the target attacking traffic observed by router R_i should be in proportion to the r_i parameter. The secondary attacking traffic simulates the traffic filtered by another filtering rule which will be selected to be replaced by the new filtering rule, therefore the volume of the secondary attacking traffic should be proportion to the l_i parameter. The target attacking traffic is simulated by flooding ping from each attacking host toward the victim host with multiple threads of the following command:

```
ping -f -s 1024 192.168.5.2 &
```

Note that the ping command is executed in the background (by specifying the "&" at the end), so that we can execute different number of threads of ping at different attacking hosts to simulate different volume of attacking traffic coming into each router.

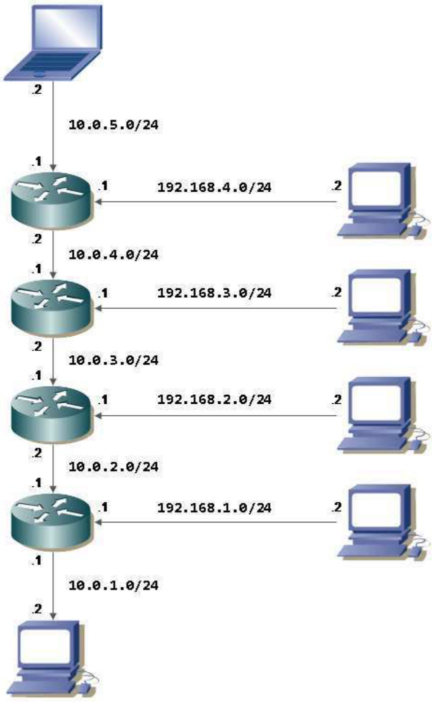


Fig. 5 Configuration of testbed network.

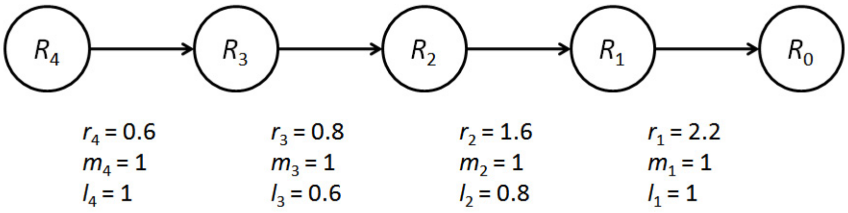


Fig. 6 Parameters used in the simulation experiments.

Moreover, in order to simulate l_i, namely the performance loss due to replacing an existing filtering rule at router R_i, we need to simulate a secondary attacking traffic coming into each router and being filtered by a different rule. When one router installs the filtering rule on the target attacking traffic, the other rule filtering the secondary attacking traffic is disabled, so as to simulate the performance loss due to installing the new rule. It would be ideal if we can connect a second host to each router to generate the secondary attacking traffic, so that the filtering of the secondary attacking traffic will be based on source address, and the comparison of the impact on legitimate traffics throughput due to the target attacking traffic and the impact due to the secondary attacking traffic will be fair. However, because of the constraint of our currently available equipment, we have to use an alternative approach in which we distinguish the target attacking traffic from the secondary attacking traffic by using the Type of Service (TOS) value [2], which can be specified as one optional field in the ping packet and can also be specified in a filtering rule. For example, to flood ping packets with TOS value 4 toward the victim 192.168.5.2, we can execute multiple threads of the following command:

```
  ping -f -s 1024 Q 4 192.168.5.2 &
```

We generated the target attacking traffic which traverses all 4 routers as type 0 (the normal type), the secondary traffic coming into R_1 as type 4, the secondary traffic coming into R_2 as type 8, the secondary traffic coming into R_3 as type 12, and the secondary traffic coming into R_4 as type 16.

Based on the parameters defined in Fig. 6, we execute 60 threads of ping at the host 192.168.4.2, 20 threads of ping at the host 192.168.3.2, 80 threads of ping at the host 192.168.2.2, and 60 threads of ping at the host 192.168.1.2, so that the volume of target attacking traffic injected into each router is proportional to r_4, r_3, r_2, and r_1. We also execute appropriate number of ping threads with TOS assigned so that the volume of secondary attacking traffic injected into each router is proportional to l_4, l_3, l_2, and l_1.

In Step 4, by default, each router installed the following set of ACL rules to filter the secondary attacking traffic:

```
  access-list 100 deny icmp any any tos X {X is a chosen
tos value}
  access-list 100 permit icmp any any
  access-list 100 permit ip host 192.168.6.2 any
```

If a router is chosen to install the filtering rule on the target attacking traffic in Step 5, the router will disable the above rules and install the following set of ACL rules:

```
  access-list 101 deny icmp any any tos normal
  access-list 101 permit icmp any any
  access-list 101 permit ip host 192.168.6.2 any
```

In Step 5, we generate legitimate traffic and test it with each possible combination of routers installing the filtering rule in order to measure the throughput of legitimate traffic of each case. The legitimate traffic is generated by executing iperf [11] client procedure on the host 192.168.6.2 which is connected to router R_4, and executing iperf server procedure on the victim host 192.168.5.2. The iperf tool can measure the amount of data transferred between the client and the server during a given period of time, and calculate the throughput of the data path accordingly.

We test all possible combinations of routers installing the filtering rule. There are totally 15 combinations to test, because there are 4 routers on the path and each router can be included or excluded in the combination except for the case in which no router installs the filtering rule. The effectiveness of our algorithm will be validated if a case with higher net saving $\Delta SAV(R_{ind},k)$ calculated by the proposed algorithm has higher throughput of legitimate traffic. Table 1 shows the solution derived from the dynamic programming based algorithm, including the best net saving and the set of routers that should install the filtering rule to get the best net saving. Table 2 shows a list of the net saving calculated by the algorithm and the measured throughput of legitimate traffic in the simulation for all possible combinations of routers installing the filtering rule.

	K				
	0	1	2	3	4
R_1	0	1.2 $\{R_1\}$	1.2 $\{R_1\}$	1.2 $\{R_1\}$	1.2 $\{R_1\}$
R_2	0	2.4 $\{R_2\}$	2.4 $\{R_2\}$	2.4 $\{R_2\}$	2.4 $\{R_2\}$
R_3	0	2.4 $\{R_2\}$	2.6 $\{R_2,R_3\}$	2.6 $\{R_2,R_3\}$	2.6 $\{R_2,R_3\}$
R_4	0	2.4 $\{R_2\}$	2.6 $\{R_2,R_3\}$	2.6 $\{R_2,R_3\}$	2.6 $\{R_2,R_3\}$

Table 1 Solution to the simulation example using the dynamic programming based algorithm.

Note that the first row in Table 2, which states "No filtering", refers to the contrast case where all four routers continue to filter only the secondary attacking traffic and do not install the filtering rule against the target attacking traffic. From Table 2, it can be seen that the measured throughput is improved when the filtering rule against the target attacking traffic is installed on any subset of the four routers in our testbed. The measured throughput of the legitimate traffic is largely consistent with the net saving calculated by the proposed algorithm. In particular, the results show that the two cases that lead to highest net saving are installing the filtering rule at routers R_2 and R_3 or installing at routers R_2 and R_4, which is consistent with the results calculated by the algorithm.

5 Conclusion

In this chapter, we present a router-based filtering technology to address the availability issues in cloud computing. We first introduced the BGP Flow-Spec rules, which provide a handy means for routers to exchange information about early filtering of overwhelming malicious traffic that could impact the availability of cloud services. Then, we presented a theoretical model and a dynamic programming based algorithm to find the best routers in a network to install the filtering rule against given malicious traffic flow, and discussed how to integrate this algorithm with cloud au-

Combinations of Routers Installing the Filtering Rule	**Net Saving Calculated by Algorithm**	**Throughput Measured by Simulations (Mb/sec)**
No filtering	0	195.2
Filtering at R_1	1.2	270.6
Filtering at R_2	2.4	303.6
Filtering at R_3	1.8	250.6
Filtering at R_4	1.4	222.2
Filtering at R_1, R_2	2	251.9
Filtering at R_1, R_3	2.2	277.2
Filtering at R_1, R_4	2	277.2
Filtering at R_2, R_3	2.6	306.7
Filtering at R_2, R_4	2.6	304.8
Filtering at R_3, R_4	1.4	247.0
Filtering at R_1, R_2, R_3	2.2	248.3
Filtering at R_1, R_2, R_4	2.2	247.8
Filtering at R_1, R_3, R_4	1.8	275.9
Filtering at R_2, R_3, R_4	2.2	300.4
Filtering at R_1, R_2, R_3, R_4	1.8	243.2

Table 2 List of net saving calculated by the algorithm and measured throughput in the simulation for all possible combinations of routers installing the filtering rule.

diting techniques. Finally, we used experimental results to validate the effectiveness and benefits of our theoretical model and algorithm.

Although we have demonstrated the effectiveness of the router-based early filtering approach, how to ensure the responsiveness of this approach remains an open challenge. It may take some time to detect an ongoing malicious attack and generate the corresponding filtering rule, and during this period of time the availability of cloud services could be seriously impacted. It is worthy to investigate how to enhance the responsiveness of the approach such that the attack can be mitigated shortly after its outbreak. Moreover, with the fast proliferation of mobile devices that can access the cloud services, it is also important to manage the scalability of the proposed approach.

Acknowledgements This work is partially supported by the NSF under Award No. CNS-0916857, by AFOSR under a Summer Faculty Fellow Program (SFFP) award and an instrumentation grant, and by AFRL under a Visiting Faculty Research Program (VFRP) award.

References

1. Andrei Agapi, Ken Birman, Robert M. Broberg, Chase Cotton, Thilo Kielmann, Martin Millnert, Rick Payne, Robert Surton, and Robbert van Renesse. Routers for the Cloud: Can the Internet Achieve 5-Nines Availability? *IEEE Internet Computing*, 15(5):72–77, 2011.

2. Philip Almquist. Type of Service in the Internet Protocol Suite. RFC 1349, July 1992.
3. Shigang Chen and Klara Nahrstedt. An overview of qualityofservice routing for the next generation high-speed networks: problems and solutions. *IEEE Networks, Special Issue on Transmission and Distribution of Digital Video*, 12(6):64–79.
4. Richard Deal. *Cisco Router Firewall Security*. Cisco Press, Indianapolis, 2005.
5. Bertrand Duvivier. Cisco BGP Security Enhancements. http://www.cisco.com/web/offer/emear/20940/presentations/Day_2_S4_SP_Security_Froum-BGP_enhancements.pdf, 2011.
6. Donna Ghosh, Venkatesh Sarangan, and Raj Acharya. Quality-of-service routing in ip networks. *IEEE Transactions on Multimedia*, 3(2):200–208.
7. Chin-Tser Huang, Keesook J. Han, and James Perretta. Automatic Selection of Routers for Placing Early Filters of Malicious Traffic. In *Proceedings of IEEE GLOBECOM 2011*, pages 1–5, 2011.
8. Prasanth Kalakota and Chin-Tser Huang. On the Benefits of Early Filtering of Botnet Unwanted Traffic. In *Proceedings of 18th International Conference on Computer Communications and Networks*, pages 1–6, 2009.
9. Pedro Marques, Nischal Sheth, Robert Raszuk, Barry Greene, Jared Mauch, and Danny McPherson. Dissemination of flow specification rules. RFC 5575, August 2009.
10. Juniper Networks. Enabling BGP to Carry Flow-Specification Routes. http://www.juniper.net/techpubs/en_US/junos10.2/topics/usageguidelines/routing-enablingbgptocarryflowspecificationroutes.html, April 2010.
11. NLANR/DAST. Iperf. http://iperf.sourceforge.net/, 2008.
12. Quagga. Quagga Routing Software Suite. http://www.nongnu.org/quagga, 2011.
13. Yakov Rekhter, Tony Li, and Susan Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.
14. Stavroula Siachalou and Leonidas Georgiadis. Efficient qos routing. In *Proceedings of IEEE INFOCOM 2003*, pages 351–367, 2003.

AFOSR Deliverables Submission Survey

Response ID: 2227 Data

1.

Report Submission Form

If you have any questions, please contact your Program Manager or Assistant Program Manager.

Air Force Office of Science and Research
875 Randolph Street
Suite 325 Room 3112
Arlington, VA 22203

1. Report Type

Final Report

4. Primary Contact E-mail
Contact email if there is a problem with the report.

shxu@cs.utsa.edu

5. Primary Contact Phone Number
Contact phone number if there is a problem with the report

2108570160

6. Organization / Institution name

University of Texas at San Antonio

Award Information

8. Grant/Contract Title
The full title of the funded effort.

IAPD: Integrated Adaptive and Proactive Defense against Stealthy Botnets

9. Grant/Contract Number
AFOSR assigned control number. It must begin with "FA9550" or "F49620".

FA9550-09-1-0165

10. Principal Investigator Name
The full name of the principal investigator on the grant or contract.

Shouhuai Xu

11. Program Manager
The AFOSR Program Manager currently assigned to the award

Dr. Robert Herklotz

Report Information - Annual Report

Report Information - Final Report

Report Information - Conference/Workshop Report

Report Information - Equipment Report

Report Information - Patent/Invention Report, DD882

Report Information - Financial Report, SF425

Report Information - STTR Status / Annual Progress Report

For an annual report, the reporting period start date is either start date of the grant, if this is the first report, or 1 day after the due date of the previous report. The end date is due date of this report.

The reporting period start and end dates are the start and end dates of the award.

21. Reporting Period Start Date

03/01/2009

22. Reporting Period End Date

11/30/2012

Report Abstract:

In the Abstract section, please list any accomplishments that have been made since the last report submission (or since the beginning of the award if this is the first report).
Please do not type "see report" here, include at least an abstract, 250 words or more, of the accomplishments mentioned in your report.

Report Abstract:

In the Abstract section, enter the Final Conference Report. This is a summary of all scientific papers presented and a list of all attendees.

Report Abstract:

In the Abstract section, enter the Final Performance Report.

The Final Performance Report will identify the acquired equipment (although it may vary from that described in your proposal) by name and associated costs. The Final Performance Report shall summarize the research or educational project for which the equipment will be used.

The patent and inventions coverage contained in Article 36, Intangible Property, of the Research Terms and Conditions does not apply to this award.

Article 15, Intangible Property, in the AFOSR Agency Specific Requirements does not apply to this award.

26. Abstract

This project studies how to combat stealthy botnets and malwares by exploring a novel framework called IAPD, which stands for “Integrated Adaptive and Proactive Defenses.'' To achieve the goal, we take a systems-and-theory methodology, meaning that on one hand, we want to build systems that can deal with stealthy attacks, and on the other hand, we want to build a theoretical and foundational understanding of botnets. Such a theoretical understanding allows us to pave the way for achieving principled modeling, management, and decision-making in cyber defense. For systems research, we have built a real-life malware behavior system called Online Malware Analysis System, which is under significant further enhancements in design and implementation towards a practical tool. For theoretical research, we have been building mathematical models for understanding and reasoning the attack-defense interactions in cyberspace. Exciting results have been made and accepted for publications at prestigious venues. We have made substantial technology transfer to AFRL, including two joint patent applications with AFRL researchers. While referring to the report for details, we highlight some of the findings as follows. --There are many bots/malwares that can evade the COTS anti-malware detection tools. Next generation cyber defense technology will have to be simultaneously behavior-based, adaptive and proactive. We have made substantial progress on these fronts. Our future studies aim to systematically characterize the distinguishing features of malwares based on our large malware database. This will guide us to design the next generation malware/bot defense technology. --In order to quantify the (in)effectiveness of cyber defense, we need a theoretical foundation. Our studies showed that for this purpose, multiple disciplines can serve as useful modeling tools, including Stochastic Processes, Dynamical Systems, Control Theory, Statistical Physics and Statistics. We will further our studies toward a unified foundation. Our envisioned foundation will become on

indispensable cornerstone of the emerging Science of Security. In particular, we found that existing mathematical/statistical/physical theories are not sufficient for cybersecurity; instead, we need to develop new theories that are guided by, and centered on, cybersecurity problems (i.e., “mathematical theories with cybersecurity meanings/significance”). --We made several conceptual advancements. For example, there are two classes of malware-based attacks: push-based and pull-based. Systematic characterization of these classes of attacks will deepen our understanding of cybersecurity. For this purpose, we introduced some core abstractions for understanding cybersecurity, including stochastic cyber attack processes, stochastic cyber attack-defense processes, cyber attack-defense dynamics. Finally, cyber attacks and cyber attack-defense interactions exhibit rich phenomena. This highlights the importance of studying the phenomenon-perspective of cybersecurity (or cybersecurity phenomena), which was not recognized until now. --Our study touched several deepest concepts in cybersecurity. For example, we showed for the first time that cyber attack traffic exhibits the so-called Long-Range Dependence, which was previously known to be exhibited by benign traffic. This hints that it is perhaps impossible to detect (stealthy) attacks based on traffic alone. This sheds lights on the detectability of cyber attacks. On the other hand, we showed that Long-Range Dependence can be exploited to predict cyber attacks at a reasonable accuracy. This sheds light on the predictability of cyber attacks. It would be of fundamental value to characterize the interaction (or trade-off) between detectability and predictability of cyber attacks.

27. Distribution Statement
This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

28. Explanation for Distribution Statement
If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

NOTE: Extra documentation is NOT required for this report. If you would like to send additional documentation, send it directly to your Program Manager or Assistant Program Manager.

30. SF298 Form
Please attach your SF298 form. A blank SF298 can be found here. Please do not spend extra effort to password protect or secure the PDF, we want to read your SF298. The maximum file size for SF298's is 50MB.

AFD-070820-035.pdf

Upload the Report Document. The maximum file size for the Report Document is 50MB.

UTSA_Shouhuai_XU_AFOSR_Grant_FA9550-09-1-0165_Final_Report.pdf

Additional Information

33. Archival Publications (published) during reporting period:

See References Section in the final report.

34. Changes in research objectives (if any):

35. Change in AFOSR Program Manager, if any:

36. Extensions granted or milestones slipped, if any:

For an STTR Status or STTR Annual Progress Report, please e-mail your program manager directly.

2. Thank You

Your report has been submitted. You should receive an email confirmation soon that it is being processed by AFOSR. Please print this page as proof of submission. Thank you.

Principal Investigator Name:	Shouhuai Xu
Primary Contact E-mail:	shxu@cs.utsa.edu
Primary Contat Phone Number:	2108570160
Grant/Contract Title:	IAPD: Integrated Adaptive and Proactive Defense against Stealthy Botnets
Grant/Contract Number:	FA9550-09-1-0165
Program Manager:	Dr. Robert Herklotz
Report Type:	Final Technical
Reporting Period Start Date:	03/01/2009
Reporting	11/30/2012

Period End Date:

Abstract:

This project studies how to combat stealthy botnets and malwares by exploring a novel framework called IAPD, which stands for “Integrated Adaptive and Proactive Defenses.'' To achieve the goal, we take a systems-and-theory methodology, meaning that on one hand, we want to build systems that can deal with stealthy attacks, and on the other hand, we want to build a theoretical and foundational understanding of botnets. Such a theoretical understanding allows us to pave the way for achieving principled modeling, management, and decision-making in cyber defense. For systems research, we have built a real-life malware behavior system called Online Malware Analysis System, which is under significant further enhancements in design and implementation towards a practical tool. For theoretical research, we have been building mathematical models for understanding and reasoning the attack-defense interactions in cyberspace. Exciting results have been made and accepted for publications at prestigious venues. We have made substantial technology transfer to AFRL, including two joint patent applications with AFRL researchers. While referring to the report for details, we highlight some of the findings as follows.

--There are many bots/malwares that can evade the COTS anti-malware detection tools. Next generation cyber defense technology will have to be simultaneously behavior-based, adaptive and proactive. We have made substantial progress on these fronts. Our future studies aim to systematically characterize the distinguishing features of malwares based on our large malware database. This will guide us to design the next generation malware/bot defense technology.

--In order to quantify the (in)effectiveness of cyber defense, we need a theoretical foundation. Our studies showed that for this purpose, multiple disciplines can serve as useful modeling tools, including Stochastic Processes, Dynamical Systems, Control Theory, Statistical Physics and Statistics. We will further our studies toward a unified foundation. Our envisioned foundation will become on indispensable cornerstone of the emerging Science of Security. In particular, we found that existing mathematical/statistical/physical theories are not sufficient for cybersecurity; instead, we need to develop new theories that are guided by, and centered on, cybersecurity problems (i.e., “mathematical theories with cybersecurity meanings/significance”).

--We made several conceptual advancements. For example, there are two classes of malware-based attacks: push-based and pull-based. Systematic characterization of these classes of attacks will deepen our understanding of cybersecurity. For this purpose, we introduced some core abstractions for understanding cybersecurity, including stochastic cyber attack processes, stochastic cyber attack-defense processes, cyber attack-defense dynamics. Finally, cyber attacks and cyber attack-defense interactions exhibit rich phenomena. This highlights the importance of studying the phenomenon-perspective of cybersecurity (or cybersecurity phenomena), which was not recognized until now.

--Our study touched several deepest concepts in cybersecurity. For example, we showed for the first time that cyber attack traffic exhibits the so-called Long-Range Dependence, which was previously known to be exhibited by benign traffic. This hints that it is perhaps impossible to detect (stealthy) attacks based on traffic alone. This sheds lights on the detectability of cyber attacks. On the other hand, we showed that Long-Range Dependence can be exploited to predict cyber attacks at a reasonable accuracy. This sheds light on the predictability of cyber attacks. It would be of fundamental value to characterize the interaction (or trade-off) between detectability and predictability of cyber attacks.

Distribution Statement: Distribution A - Approved for Public Release

SF298 Form: 11-c74f145de0f87abb38c7a7b77a2b192d_AFD-070820-035.pdf

Report Document: 54-28f0e65b5c7287e137eccf1bbdb86170_UTSA_Shouhuai_XU_AFOSR_Grant_FA9550-09-1-0165_Final_Report.pdf

Archival Publications: See References Section in the final report.

Changes in Research objectives:

Change in AFOSR Program Manager, if any:

Extensions granted or milestones slipped, if any: